UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Desenvolvimento de Solução Microsoft para Instituição Financeira

Daniel Salvador Lourenço

Mestrado em Engenharia Informática
Especialização em Engenharia de Software

Trabalho de Projeto orientado por:
Prof. Doutor Paulo Jorge Cunha Vaz Dias Urbano

2019

Agradecimentos

A realização de uma dissertação de mestrado é um longo caminho, do qual não pode ser realizado sem o apoio que nos rodeiam. Espero nesta secção agradecer a todos, que de forma direta ou indireta, me apoiaram durante a realização deste projeto.

Primeiramente gostaria de agradecer ao professor Doutor Paulo Urbano, por toda a sua disponibilidade e, principalmente, por toda a paciência e ajuda ao longo deste percurso.

Em segundo lugar agradecer ao Ricardo Castanheira pela oportunidade que me deu em fazer este projeto na Unipartner. Obrigado pela forma como me recebeu na equipa, pela presença e apoio constantes.

Um agradecimento especial ao João Vasco Ferreira, que foi essencial à minha adaptação a nível técnica e profissional. Por toda a presença e apoio, por todas as vezes em que me "deu na cabeça" e principalmente pela amizade e disponibilidade sempre presentes.

Um grande obrigado a toda a equipa da Unipartner, pela forma como fui acolhido e integrado no grupo, pelos momentos vividos com todos e pela excelência que sempre demonstra e transmite aos outros.

Para a minha família não chegam todos os agradecimentos do mundo, nomeadamente aos meus pais, pela educação que me deram e por me tornarem a pessoa que sou hoje. À minha irmã, por todo o carinho e amizade demonstrados e pelas brincadeiras de sempre. São vocês a minha motivação diária, um sincero obrigado!

Por fim, não podia deixar de gratificar a pessoa que me soube aturar nos momentos de maior frustração, que me apoiou, que estava lá nos bons momentos, mas principalmente que me levantou nos momentos mais complicados. Espero que saibas o quanto valorizo todos estes gestos, obrigado Ekaterina!

Quero demonstrar também a minha gratidão a todos os meus amigos e àqueles que de uma maneira ou de outra contribuíram para a concretização desta dissertação.

Aos meus pais, à minha irmã.

Resumo

Esta tese foi escrita no âmbito de um estágio curricular com o tema "Desenvolvimento de Solução Microsoft para Instituição Financeira", realizado ao serviço da Unipartner IT Services S.A. numa seguradora.

O projeto é enquadrado no sistema informático da seguradora, tendo este por base uma Solução Multicanal, Multi-plataforma e Multi-browser.

A Solução é refletida numa arquitetura em camadas, que comunicam entre si através de um *bus* de eventos (sistema onde são publicados ou consumidos serviços, de forma a facilitar a comunicação entre componentes) que fornece serviços a ser consumidos entre as diversas camadas.

O projeto de engenharia informática realizado durante o estágio curricular incide principalmente sobre a camada de *Middleware*. Esta assenta na *framework* .NET, cujos desenvolvimentos são realizados na linguagem C#. A comunicação com a camada das bases de dados (que correm em Microsoft SQL Server) é realizada através de *scripts* escritos na linguagem T-SQL.

No *Middleware*, o meu trabalho centrou-se no módulo do Motor de Impressão, sendo este responsável por gerar documentos (pdf) pedidos pelo utilizador. Este recolhe todas as informações a serem apresentadas no documento final, trata os dados e chama depois um serviço no qual é dado um XML de *input* e é obtido o pdf final. Este tem também independência para expedir os ficheiros pelos diversos meios (FAX, Email e Carta).

Este projeto surgiu no âmbito da migração da ferramenta que gera os outputs da empresa, sendo para isso fazer os desenvolvimentos necessários ao componente do MI de forma a que este se adapte à nova ferramenta.

Palavras-chave: Microsoft, *Middleware*, *bus* de eventos, motor de impressão

Abstract

This Theses was written as part of a curriculum internship with the theme ”Development of Microsoft Solution for Financial Institution”, carried out at the service of Unipartner IT Services S.A. at an insurance company.

The project is framed in the insurer's IT system. This system is based on a Multichannel, Multi-platform and Multi-browser Solution.

The Solution is reflected in a layered architecture, where the communication between the layers goes through an event bus (a system where events are published or consumed to facilitate communication between components) that provides services to be consumed between the various layers.

The computer engineering project carried out during the curricular internship focuses mainly on the Middleware layer. This layer is built on the .NET Framework and the developers code using C# language. The communication with the database layer (running on a Microsoft SQL Server) is accomplished using scrips (Stored Procedures) that are written in T-SQL.

In the Middleware, my work focused on the print engine module of this layer. The print engine is responsible for generating documents (pdf) requested by the user. It collects all information to be presented in the final document, process the data and then calls a service in which an input XML is given, and the final pdf is generated. This print engine also has independence to send the files through many ways (FAX, Email and Letter).

This project arose within the scope of the migration of the tool that generates the outputs of the company, making the necessary developments to the print engine component so that it adapts to the new tool.

Keywords: Microsoft, *Middleware*, event bus, print engine

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o crescimento do mundo digital, cada vez mais empresas procuram impulsionar-se no mercado através de tecnologias disruptivas para poder combater a grande concorrência existente num mercado que é cada vez mais complexo. O setor de seguros não representa exceção, portanto é necessário otimizar os custos, aumentando a eficiência e diminuindo o erro humano. Nas companhias de seguros, grande parte do seu fluxo de negócio consiste na geração de documentos, pois são essenciais aos seus clientes. Destes destacam-se Declarações de Seguro (comprovativo de que o cliente está assegurado), Condições Particulares (descrição de todas as condições relativas ao seguro), Cartas Verdes (comprovativo que um veículo tem seguro), Recibos, Faturas, entre muitos outros. Os documentos tomam assim um papel importante na comunicação com o cliente bem como na aplicação e divulgação das condições do negócio. Estes documentos, dependendo do tipo, podem conter centenas de páginas, sendo que incluem informação específica do contrato bem como do próprio segurado.

No passado, esses documentos de seguro eram criados de diversas formas como escrevendo cartas de forma livre, tendo folhetos pré-impressos, editando modelos previamente definidos, etc. Posteriormente os documentos criados eram agrupados e enviados para o segurado. Todo este processo era feito manualmente, exigindo uma grande mão de obra na formulação e revisão dos documentos e estando sujeito a riscos associados ao erro humano.

A geração automática de documentos permite que se criem modelos que sirvam de base para o documento. Posteriormente, um *software* preenche esse modelo com os dados do cliente e do contrato ou simulação que lhe são fornecidos. Desta forma a geração de documentos é feita de forma automática, sendo o seu desenvolvimento e manutenção muito mais fáceis e rápidos.

Na companhia onde se realizou este projeto, o sistema de automatização de documentos é composto por dois componentes, sendo eles o Motor de Impressão (MI) e a ferramenta de geração de documentos (Adobe Output Designer - AOD). O MI é responsável por obter e tratar toda a informação necessária para que se possa gerar um documento.

Posteriormente, o AOD contém modelos de documentos com a informação fixa, ou seja, dados comuns a todos os documentos desse modelo, sendo depois os dados variáveis preenchidos com a informação proveniente do MI na hora da geração de documentos (visível na figura 1.1). Cada modelo tem associado um id único que o identifica para que, na hora da obtenção de documentos, o MI possa informar a ferramenta de geração de PDFs de qual o modelo a ser utilizado. De seguida, o AOD gera o PDF final a ser devolvido ao cliente. No caso dos documentos relativos a uma apólice, existe a possibilidade de serem impressos em conjunto, sendo que neste caso o MI obtém os documentos de forma cíclica e no final utiliza outro componente para os agrupar num único PDF. Por fim, é guardado um registo de cada documento obtido para que possa existir um histórico da geração de cada um dos modelos.

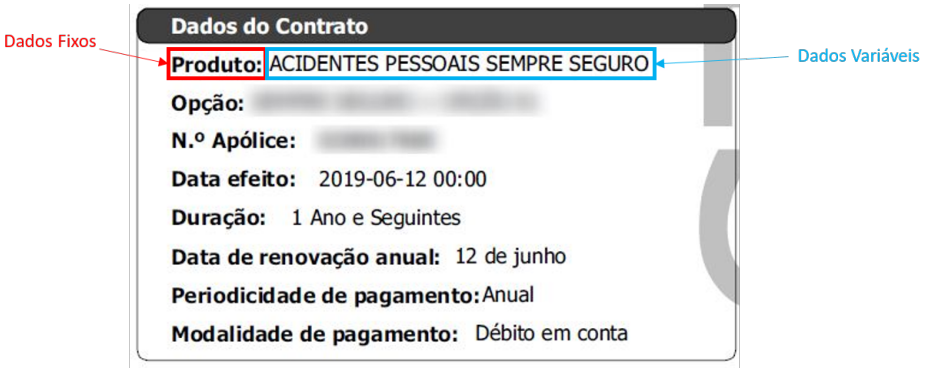

Figura 1.1: Exemplo de dados fixos e variáveis num documentos

O MI tem ainda capacidade para enviar os documentos pelo meio desejado pelo utilizador, podendo ser este carta, FAX, email ou impressão (apresentação do PDF para *download*). No caso dos emails, os documentos gerados são inseridos num email cujo corpo e assunto se encontram previamente guardados nas bases de dados da companhia. Esses, por sua vez, são definidos por uma variável enviada pela camada de interface com o utilizador, permitindo assim que o email seja diferente consoante a ação que está a executar no momento (um email com os documentos pedidos para envio pelo cliente ao consultar uma apólice deve ser diferente daquele que acompanha o envio de uma simulação de um contrato).

O MI é um componente que faz parte do Sistema de Informação da companhia. Este é composto por vários componentes, sendo a sua comunicação baseada numa Arquitetura Orientada a Serviços (SOA). Assim, quando um componente pretende aceder a uma função de outro componente, fá-lo através do serviço disponibilizado para o efeito. Neste caso, é utilizado o formato XML como forma de estruturar o *input* e o *output* do serviço. Este XML é composto por *tags* que contêm toda a informação necessária a esta comunicação.

1.1 Motivação

A obtenção de documentos é essencial no negócio da companhia e no dia-a-dia dos seus mediadores. É por isso necessário que o sistema de geração de documentos esteja em constante atualização de forma a que não se torne obsoleto.

No caso deste projeto, a ferramenta de geração de PDFs (Adobe Output Designer - AOD) encontrava-se desatualizada e foi identificada uma ferramenta da HP, o HPEstream (HPX) de forma a garantir esta atualização e o acompanhamento da evolução do mercado.

Quando existe a necessidade de alterar um determinado documento, o pedido é feito à equipa que desenvolve o MI. Esta é responsável por fazer as alterações necessárias. De seguida deve enviar um email para a equipa que desenvolve os modelos na ferramenta de geração de documentos de forma a informá-los como estas alterações se refletem no XML que é enviado com a informação necessária para gerar o respetivo ao documento.

1.1.1 Adobe Output Designer

Para a geração de documentos da empresa, era utilizada uma ferramenta fornecida pela Adobe, o Adobe Output Designer (AOD). Com o passar tempo esta ferramenta veio a mostrar-se muito limitada, tendo já sido descontinuada pelo próprio fornecedor (Adobe).

Por exemplo, no caso representado na figura 1.2, as *tags* relativas ao "Nome" e "Morada" têm de ser duplicadas no ficheiro XML pois o AOD apenas permite que o seu uso seja feito uma vez. Esta limitação reflete-se na maioria dos documentos obtidos através do MI, pois os campos de Dados de Cliente e de Morada mantém-se ao longo de todos os documentos, levando assim à duplicação desnecessária de informação no XML por parte do MI, tendo um maior custo de desenvolvimento.

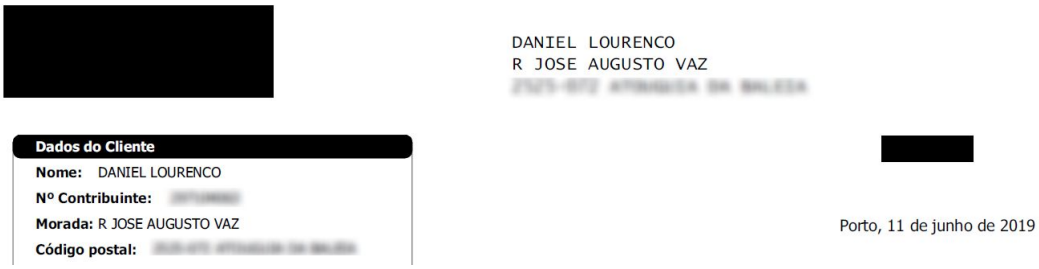

Figura 1.2: Exemplo de caso com dados repetidos

Por outro lado, esta ferramenta é restrita relativamente à ordem do XML face ao desenho do modelo. Por exemplo, no caso da figura 1.2, a *tag* que contém a informação relativa ao campo onde está colocado o "Nome" tem de vir primeiro no ficheiro XML para poder estar representado antes do "Nº Contribuinte". Caso isto não se verifique, o documento ou não é gerado ou os campos em questão não vêm representados.

Outra limitação encontra-se na criação de modelos distintos para ficheiros XML cuja estrutura é praticamente idêntica. Como exemplo podemos destacar os documentos de

cariz financeiro como uma fatura ou um aviso (Figura 1.3) que têm poucas diferenças na sua estrutura e o AOD não permite a partilha do mesmo modelo, sendo assim necessário criar um para cada um destes documentos, aumentado muito o tempo de desenvolvimento. Embora o AOD não permita a partilha de informação, esta é agrupada pelo MI, pois a estrutura de classes do C#, devido à sua arquitetura ser orientada a objetos, este agrupamento é feito de forma hierarquizada, ou seja, é agrupada a informação comum, e posteriormente detalhada consoante o modelo a utilizar.

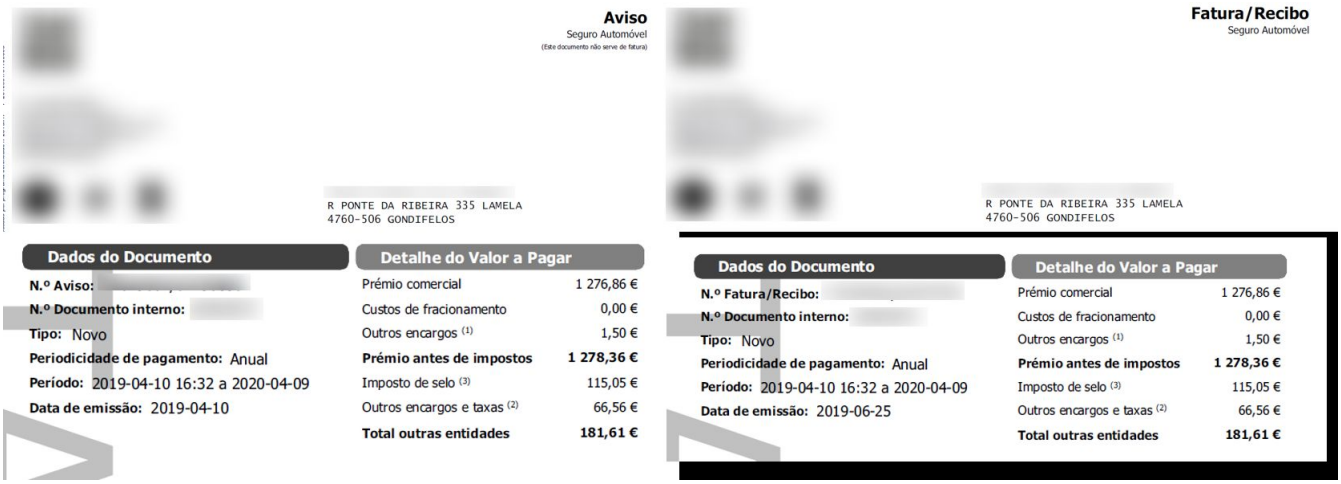

Figura 1.3: Semelhanças de documentos financeiros

O facto da partilha de informação relativa aos desenvolvimentos entre a equipa responsável pelo MI e a que trata dos desenhos dos modelos ser feita via Email leva a que seja induzido erro humano nessa comunicação como o envio de *tags* por parte do MI que ainda não foram mapeadas para o respetivo modelo. O AOD é muito restrito relativamente a isto sendo que caso o XML contenha uma *tag* que não esteja a ser utilizada, o documento final não é gerado.

O AOD apresenta ainda uma grande limitação a nível de segurança. Os servidores onde a ferramenta se encontra em execução contém um sistema operativo desatualizado devido à antiguidade do AOD, o que pode resultar em graves problemas a nível de segurança para a companhia.

1.1.2 HP Exstream

Em alternativa foi identificada uma nova ferramenta de geração de documentos, o HP Exstream (HPX). Esta traz grandes melhorias principalmente a nível de desenho de modelos, sendo este mais fácil e rápido do que no AOD.

Relativamente à ferramenta antiga, o HPX oferece as seguintes vantagens:

- Permite que o mesmo modelo esteja preparado para processar informação de ficheiros XML com diferentes. Assim, é possível agrupar documentos com a mesma

estrutura num único modelo e estes serem gerados consoante a informação presente no XML.

- Tem um modelo base com as informações comuns a todos os documentos. No caso do cabeçalho e o rodapé, estes são utilizados em todos os modelos pelo que podem estar neste modelo base de forma a que não seja necessária a sua criação cada vez que se desenha um novo modelo.
- Reutiliza componentes ao longo de diversos modelos. Por exemplo os dados do endereço de morada são utilizados em quase todos os documentos e vêm sempre na mesma estrutura nos diferentes ficheiros de XML recebidos. Neste é criado um componente para essa estrutura. Este pode assim ser utilizado nos modelos de documentos que necessitam dos dados da morada.
- Utiliza regras para certos campos do modelo. No caso de protocolos que existam com empresas de seguros parceiros, o logótipo que vem no documento pode ser customizado consoante o protocolo e/ou o parceiro em questão.
- Cria variáveis comuns a todos os modelos. Para as variáveis cuja *tag* é a mesma nos diversos ficheiros XML, é possível que se crie um dicionário de forma a que esta seja utilizada de uma forma mais prática nos diversos documentos.
- Permite que sejam enviados mais de que um documento por ficheiro XML. Quando se pretende gerar vários documentos relativo ao mesmo modelo, estes podem ser assim enviados no mesmo ficheiro XML.

1.2 Objetivos

Este projeto tem como finalidade **adotar o HP Exstream como ferramenta de desenho e geração de todos os documentos institucionais**, terminando com o uso do Adobe Output Designer. Na companhia, existem outras formas de obtenção de documentos institucionais sem ser o MI sendo que este projeto se reflete apenas no desenvolvimentos relativos ao MI e ao conjunto de documentos que este obtém.

Com a migração é necessário **adaptar o XML de acordo com as especificações do HPX e aproveitar a migração para uniformizar a sua estrutura**.

De forma a garantir a **coexistência de ambas ferramentas em Ambiente de Produção durante migração** o projeto foi dividido em várias etapas. Isto permite que a migração seja feita de forma gradual, não causando tanto impacto no utilizador, pois em caso de indução de erros durante os desenvolvimentos é assim possível que estes sejam corrigidos de forma controlada.

Por outro lado, é preciso **documentar as alterações** de forma a que a informação relativa ao projeto seja mantida e, quando for necessário corrigir ou perceber o que foi

alterado, seja percetível através da sua documentação. Desta forma não é necessário que seja a pessoa que implementou o projeto a fazer a alteração e sem que haja perda o conhecimento do desenvolvimento realizado.

No decorrer do projeto, e de forma a garantir uma maior rapidez e eficiência, é necessário **coordenar os meus desenvolvimentos com o setor que desenvolve os modelos no HPX** de forma a que ambas as equipas possam estar sintonizadas relativamente às mudanças no XML e a sua disponibilização. Como a comunicação de alterações é feita via Email, é necessário informar a equipa que gere os modelos para que esta tenha conhecimento caso os desenvolvimentos tiverem terminados ou se mude algo na estrutura do XML. O mesmo acontece no sentido contrário, se for necessário acrescentar alguma *tag* ou informação ao ficheiro XML de um determinado modelo.

Por fim, para que seja garantida a qualidade do projeto, será necessário **fazer testes unitários** a todos os documentos desenvolvidos bem como **acompanhar a equipa de testes da companhia durante a fase de testes de qualidade**.

1.3 Contribuições

Com o decorrer do projeto, foi possível cumprir o objetivo principal, sendo neste momento o HPX a ferramenta de desenvolvimentos e desenho de documentos padrão, estando todo o MI adaptado para que os documentos atuais e novos sejam desenvolvidos nesta nova ferramenta.

Com as minhas alterações, o XML enviado pelo MI com a informação relativa à geração dos documentos passou a ser uniforme. A estrutura dos documentos financeiros tornou-se híbrida entre ferramentas, ou seja, o seu XML manteve-se pois a sua estrutura já era idêntica. Para os outros documentos, passei para a estrutura LstDocumento/Documento, para que seja enviada a mesma estrutura para o HPX, facilitando assim o desenvolvimento dos modelos na nova ferramenta. Os campos em que a sua informação ficava desajustada ou desalinhada com a migração foram adaptados. Os *bugs* encontrados durante a migração para o HPX também foram corrigidos.

No decorrer da migração, para cada modelo fui validando o registo de impressão do MI de forma a perceber a utilização de cada um dos documentos. Aqueles que não tivessem a ser obtidos por parte dos utilizadores, eram descontinuados, não sendo migrados para a nova ferramenta.

Este projeto permitiu que atualizasse a documentação do Motor de Impressão. Assim, quando for necessário alguém aprender o funcionamento do MI, tem nesta documentação uma forte ajuda em perceber todo este componente. Isto facilita a integração de novos membros na equipa, pois têm uma maior perceção de todos os fluxos e da informação que é guardada nas bases de dados. Para que no futuro seja fácil perceber o que foi alterado no projeto foram documentadas as alterações realizadas.

De forma a validar os meus desenvolvimentos, realizei testes unitários e integrados à impressão de todos os documentos. Isto contribuiu para reportar erros também a nível do desenho.

Finalmente acompanhei a equipa de testes, contribuindo com informação para os seus casos de teste, esclarecendo as devidas dúvidas relativas aos meus desenvolvimentos, de onde era obtida a informação e se a mesma se encontrava correta.

Apesar da migração incluir documentos gerados por várias vias entre ferramentas na companhia, as minhas alterações refletiram-se apenas a documentos que são gerados através do Motor de Impressão.

1.4 Estrutura do documento

Este documento encontra-se organizado da seguinte forma:

- No capítulo 2 explana-se os termos e definições específicos desta área da Engenharia Informática e que se relacionam diretamente e são utilizados ao longo do documento.
- No capítulo 3 descreve-se todo o funcionamento do Motor de Impressão, onde este se insere na arquitetura do sistema e o modelo de dados que este utiliza no seu normal funcionamento.
- No capítulo 4 são descritos os pontos essenciais à implementação do projeto e como esta foi realizada.
- No capítulo 5 descreve-se os resultados do decorrer do projeto a nível de documentos migrados / descontinuados, bem como a sua discussão.
- No capítulo 6 conclui-se a melhoria existente com o decorrer do projeto. Além disso, reflete-se sobre o que fica para melhorar após a finalização deste projeto.

Capítulo 2

Trabalho Relacionado

O Sistema de Informação da companhia, na qual o Motor de Impressão se insere, é baseada numa Arquitetura Orientada a Serviços (SOA). Esta apresenta uma forma de comunicação entre dois componentes, sendo que para definir essa comunicação utiliza o protocolo SOAP. Este protocolo, por sua vez, é baseado em XML e, tratando-se de serviços *web*, é utilizado o WSDL (variante do XML). No caso da comunicação entre o MI e a ferramenta de geração de documentos, o ficheiro XML com a informação relativa ao documento está contido num serviço *web*.

De forma a adaptar o conteúdo dos serviços ao esquema de classes do MI, este recorre ao uso de *wrappers* de forma a consumir e encapsular os serviços e adaptá-lo à arquitetura em que este consiste. Assim, é possível que seja abstraído o serviço e a sua chamada na manutenção e desenvolvimento do MI.

O MI está implementado na linguagem C#, sendo que esta tecnologia utiliza módulos (DLLs), dividindo o código e os dados consoante a sua função.

Todos estes termos mais complexos e úteis para compreender melhor o documento são esclarecidos ao longo deste capítulo.

2.1 Arquitetura orientada a serviços

Channabasavaiah e Holley [5] definem SOA como uma arquitetura aplicacional na qual todas as funções são definidas como serviços com interfaces invocáveis. Estas funções podem ser chamadas em sequência formando assim processos de negócios. A sua definição inclui os seguintes componentes:

- Todas as funções são definidas como serviços. Independentemente da sua complexidade, todos as funções que compõem a aplicação devem poder ser chamadas através de serviços.
- Todos os serviços são independentes. Eles operam como se fossem uma "*black box*", ou seja, os componentes externos aos serviços não se interessam pela forma como estes executam a sua função, mas apenas com os seus resultados.

- Na sua generalidade, as interfaces são invocáveis. Isto é, a nível arquitetural, é irrelevante se estas são locais (fazem parte do sistema) ou remotas (externas ao sistema). Assim, são abstraídas as questões relativas ao esquema ou protocolo de conexão utilizado de modo a efetuar a chamada ou quais os componentes infraestruturais necessários para que a conexão seja possível.

Numa SOA, a interface é a chave e o foco da aplicação que é chamada. Esta define o *input* necessário e a natureza do resultado. Isso significa que define a natureza do serviço e não a tecnologia que é utilizada para imprementá-lo. O sistema deve efetuar e gerir a chamada ao serviço e não a aplicação que o implementa. Essa função permite duas caraterísticas: primeiro, que os serviços sejam verdadeiramente independentes e, segundo, que estes possam ser geridos e mantidos de forma independente do sistema que os chama.

2.2 *Simple Object Access Protocol*

SOAP é um protocolo web de nível aplicacional que permite a troca de informação em ambientes descentralizados de uma forma mais simples. É baseado em XML e pode ser combinado com diversos outros protocolos web. Este fornece um vocabulário de comunicação, definindo a estrutura de comunicação, conteúdo e sintaxe. [8]

O protocolo inclui uma estrutura de dados que codifica, como um pedido SOAP, o nome do objeto de interesse, de forma a que este seja executado remotamente com os parâmetros de input dados. Do lado do servidor, este pedido é descompactado, sendo posteriormente executado localmente o método desejado com os parâmetros de entrada e retornado o resultado da invocação do objeto através do mesmo protocolo. Este é um protocolo sem estado, ou seja, o tempo de vida de cada objeto é apenas válido por cada chamada ou método e é recriado a cada chamada do objeto.[10]

No caso de aplicações web, o SOAP torna-se muito vantajoso, pois neste tipo de aplicações é cada vez mais comum a troca de dados sobre o protocolo IP, sendo assim possível a comunicação entre aplicações distintas com diferentes tecnologias, sem ser necessário a utilização de um protocolo proprietário e permitindo assim a uniformização na Internet. [8]

2.3 Serviços *Web*

Papazoglou [12] define serviços como elementos computacionais independentes de plataforma que podem ser implementados de uma forma rápida e de baixo custo para sistemas distribuídos. Os serviços executam funções, sendo que o componente que os consome não necessita de saber a lógica de negócio em si contida, utilizando linguagens e protocolos padrão (baseados em XML).

Para Fensel e Bussler [7] os Serviços *Web* conectam computadores e dispositivos entre si usando a Internet de forma a trocar dados e a combiná-los de novas maneiras. Os serviços podem ser definidos como objetos de *software*, podendo assim ser conetados pela Internet utilizando protocolos padrão, e assim executar funções ou processos de negócio. A chave dos Serviços *Web* é o uso de componentes de *software* reutilizáveis acoplados de forma simples. Isso tem implicações em termos técnicos e comerciais. O *software* pode assim ser entregue e pago como fluxos de serviços, em vez de produtos embalados. É possível obter interoperabilidade automática entre sistemas para realizar tarefas de negócios. Os serviços de negócio podem ser completamente descentralizados e distribuídos pela Internet e acedidos por uma ampla variedade de dispositivos de comunicação. As empresas podem libertar-se do fardo da integração complexa, lenta e cara de *software* e concentrar-se no valor das suas ofertas e tarefas críticas.

2.4 *Wrapper* de um Serviço

Um *wrapper* de um serviço Web é um método usado para permitir que um software existente interaja num ambiente diferente do originalmente pretendido. [2] Já Camarinha-Matos et al.[4] referem-se a um *wrapper* de invocação de um serviço como um componente que descreve a interface de programação do serviço, ou seja, os métodos disponíveis, os seus parâmetros e valores de retorno, etc. Este atua como um *proxy* para o serviço real, proporcionando o acesso transparente a este, ocultando os detalhes da implementação, como a invocação remota ou os mecanismos de segurança e comunicação.

Figura 2.1: Comunicação entre componentes de tecnologias diferentes através de um Wrapper e um Serviço

2.5 *Web Service Description Language*

Em Christensen et al.[6], WSDL é descrito como a linguagem padrão para descrever serviços web e consiste num tipo de formato XML que descreve serviços de um conjunto de *endpoints* (entidade de comunicação capaz de realizar interações), bem como as mensagens e o fluxo de informação. Nesta linguagem são descritas as operações e as mensagens de uma forma abstrata e, em seguida, vinculadas a um protocolo de rede concreto e formatadas de acordo com o seu *endpoint*.

O WSDL permite assim que os serviços expostos sejam descritos de uma forma estruturada, para que o utilizador tenha uma melhor especificação do serviço e que dados podem ser obtidos através deste. Assim, o programador pode programar o seu sistema sabendo o que necessita caso o mesmo vá consumir o um serviço descrito em WSDL.

2.6 XML

Michael Klein [9] define XML (eXtensible Markup Language) como uma especificação para documentos legíveis por um computador. Marcação (*Markup*) significa que determinadas sequências de caracteres no documento contêm informações indicando o papel do conteúdo do documento. A marcação descreve o *layout* de dados do documento e a estrutura lógica e torna as informações auto-descritivas, assumindo a forma de palavras entre sinais de maior, menor, chamados de tags - por exemplo, <nome> ou <h1>. Nesse aspeto, o XML é muito parecido com a linguagem HTML, bem conhecida.

No entanto, ser extensível é a principal característica do XML e o que o distingue das outras linguagens do género. O XML é por isso uma metalinguagem, ou seja, uma forma padrão de representar outras linguagens. Não tem um vocabulário próprio, apenas fornece um formato, tornando-o assim como uma linguagem universal. Através do XML é possível definir linguagens de marcação personalizadas para diversos tipos de documentos. Existem por isso muitas linguagens que não são mais que uma derivação do XML mas com uma sintaxe própria. Entrando em detalhe, com o XML é possível criar entidades e por consequência criar dependências, grupos e associações de entidades. Na figura 2.2 podemos ver um exemplo disto, em que temos a entidade 'pessoa' que é composta pelas sub-entidades 'nome', 'idade' e 'profissão'.

```
<?xml version="1.0" encoding="UTF-8"?>
    <pessoa>
        <nome>Miguel</nome>
        <idade>46</idade>
        <profissao>Policia</profissao>
    </pessoa>
```

Figura 2.2: Exemplo de XML: entidade pessoa

O XML não implica desta forma uma interpretação específica. É claro que, devido aos nomes utilizados nas *tags*, o exemplo da figura 2.2 pareça óbvio se interpretado por humanos, mas não está especificada a forma como deve ser interpretado. Uma forma possível de interpretar é que existe a entidade pessoa e as sub-entidades nome, idade e profissão com os valores "Miguel", "46"e "Policia", respetivamente.

O facto de ser tão geral acaba por ser uma força e uma fraqueza do XML, uma vez que se pode codificar todos os tipos de dados e estruturas através dele de forma não ambígua, mas o XML não especifica a semântica, logo, quando existe troca de dados, ambas as partes devem concordar antecipadamente sobre o vocabulário, uso e significado.

2.7 DML, DDL e SP

A interação com as bases de dados da companhia é feita de três formas: DML, DDL e SPs. As *queries* do tipo DML são utilizadas para trabalhar os dados, sendo neste grupo incluídas as que recolhem dados, inserem linhas numa tabela, modificam valores e eliminam linhas [13]. As DDL consistem naquelas que são usadas para criação e gestão de objetos numa base de dados, podendo ser usadas para criar, modificar ou eliminar bases de dados, tabelas, índices, vistas, *stored procedures*, e outros objetos [13]. *Stored Procedures* são rotinas executadas do lado do servidor. Estas permitem pôr a lógica do acesso aos dados do lado do servidor, permitindo uma maior performance do sistema. Têm várias funções como alterar dados em tabelas, modificar definições de objetos ou servir de consulta entre várias tabelas de uma base de dados [3].

2.8 *Dynamic-link library*

Quando um produto ou sistema software é desenvolvido através do uso de tecnologia Microsoft é muito comum o uso de DLLs de forma a agrupar o código em módulos, reduzindo assim o tempo de carregar e correr o software e o espaço em disco.

A Microsoft [11] define DLL como uma biblioteca que contém código e dados e que pode ser utilizada por um ou mais programas ao mesmo tempo. Assim são poupados recursos, pois se vários programas necessitarem de utilizar uma parte de código e dados semelhantes, podem carregar e utilizar a mesma DLL poupando assim espaço em disco.

Através do uso de DLLs, os programas são assim modularizados e separados em diversos componentes. Esta técnica permite que cada módulo seja carregado apenas em tempo de execução, sendo assim carregado para memória apenas quando é necessário e, contendo apenas uma parte do código, este é carregado de forma mais rápida, aumentando assim a sua eficiência.

Outra vantagem do uso de DLLs é o momento de atualizar o software pois, neste caso, não existe a necessidade de disponibilizar todo o software mas sim apenas aqueles módulos que foram alterados. Adicionalmente se vários programas utilizarem a mesma DLL diminui o tempo que necessitaríamos para corrigir os dois, sendo apenas necessária assim a correção do módulo em comum.

Capítulo 3

Motor de Impressão

Este capítulo centra-se no componente onde está o foco deste projeto, o Motor de Impressão. Nesse sentido é descrita a arquitetura do Plataforma Web, de forma ao leitor perceber onde se situa o MI. De seguida é descrito o MI, bem como a sua estrutura, arquitetura e modelo de dados. Assim, é possível perceber o funcionamento deste componente e onde este se insere no Sistema de Informação para que seja mais fácil a compreensão das mudanças realizadas com o decorrer do projeto. Por fim são descritos os diversos ambientes integrados onde é disponibilizado o *software*.

Inicialmente, e de modo a perceber melhor o sistema e o Motor de impressão (MI), foram realizadas simulações e emissões de apólices. Assim, foi possível adquirir uma maior famialização principalmente com o fluxo do MI e assim perceber em que produtos e casos específicos se podem obter os documentos.

De forma a perceber melhor o fluxo de informação e chamadas a métodos, utilizei o *software* interno de análise de *logs* que os representa em árvore. Este *software* permite assim perceber o o funcionamento MI e a forma como este obtém a informação e interage com os outros componentes do sistema.

3.1 Arquitetura da Plataforma Web

A plataforma *web* da companhia é baseada numa arquitetura em camadas. Esta plataforma responde a pedidos de diferentes canais (para domínios diferentes da companhia, utilizados por utilizadores de diferentes hierarquias, são necessários comportamentos distintos), sendo que dependendo do canal que faz o pedido ao sistema, a sua interface e funcionalidades são diferentes, bem como a resposta ao pedido. Para a comunicação entre camadas é utilizada uma arquitetura orientada a serviços (SOA), existindo para isso um componente que faz a integração desses mesmos serviços entre as diferentes tecnologias (EAI).

O sistema representado na figura 3.1 divide-se assim em três camadas:

- Apresentação: Centra-se na interação com o utilizador, contendo por isso todas as

tecnologias utilizadas na construção de páginas com esse objetivo. Esta camada reflete-se em três componentes, sendo eles Liferay Gestão de Interfaces (LGI), OutSystems e Camada de Gestão de Interfaces (CGI). O LGI assenta no uso de tecnologia Java sendo utilizado o projeto *open source* LifeRay[1] como *framework*. O componente Outsystems, por sua vez, é baseado na própria tecnologia Outsystems. Estes dois componentes, LGI e Outsystems, acedem à camada de lógica de negócio através de *serviços* disponibilizados pelo EAI devido à diferença de tecnologia para com o componente *Middleware*. Em relação ao CGI, este assenta em tecnologia Microsoft, baseado em ASP.NET, podendo por isso consumir diretamente os serviços do *Middleware* e, dispensando assim a intermediação do EAI, visto que estão assentes em tecnologias compatíveis.

- Lógica de Negócio: É nesta camada onde está toda a lógica de negócio e são feitas a abstração na obtenção de dados e a integração entre tecnologias. Esta camada divide-se em dois grandes componentes: EAI e *Middleware*.

  O EAI tem como principal função a integração entre tecnologias de forma a que estas possam comunicar entre si, quebrando o paradigma das diferenças de linguagem ou até mesmo de implementação. Assim, o componente que utiliza um determinado serviço apenas se preocupa com a informação que obtém deste e não de toda a lógica que leva aquele resultado. Os serviços disponibilizados têm por base o protocolo *Simple Object Access Protocol* (SOAP), em que a comunicação é feita utilizando a linguagem XML e para a definição desses mesmos serviços é usada a linguagem WSDL.

  O *Middleware* disponibiliza serviços ao EAI de forma a expor o seu software e é responsável pela parte de lógica de negócio da solução. O acesso à base de dados é feito de duas formas. Caso as bases de dados sejam geridas pelo próprio *Middleware*, o acesso é feito de forma direta, pois a tecnologia entre os dois componentes é a mesma (Microsoft), sendo aí utilizadas *stored procedures* (rotinas de acesso a bases de dados que podem ser executadas do lado do servidor). Por outro lado, as bases de dados com informações relativas a apólices são acedidas através de *stored procedures*, caso assentem na mesma tecnologia, ou através de serviços fornecidos pelo EAI, nos casos em que a tecnologia difere. A linguagem utilizada é o c#, estando esta integrada na *framework* .NET da Microsoft, sendo utilizada a versão 4.5.

- *Back-End*: camada de acesso aos dados correspondente às bases de dados utilizadas pela camada de Lógica de Negócio. Esta subdivide-se em dois grupos: aquelas que são geridas pelo próprio *Middleware* e as geridas pela equipa Core de negócio (CENTRAL). As primeiras são baseadas na tecnologia Microsoft SQL Server em que a interação com a base de dados é feita através de *scripting* e é utilizado o

Transact-SQL, contendo estas principalmente parametrizações e informações relativas a simulações. As segundas são geridas pelo Central e utilizam a tecnologia COGEN. Estas contêm todas as informações relativas a apólices.

Paralelamente é utilizado o SalesForce (SF), sendo este um sistema de CRM disponibilizado na *Cloud*. O SF nesta companhia é usado para fazer a gestão de clientes e integra outros sistemas que permitem fazer gestão de sinistros, criar simulações, consultar documentos em arquivo, etc. A integração com o sistema da plataforma web é feita através do EAI.

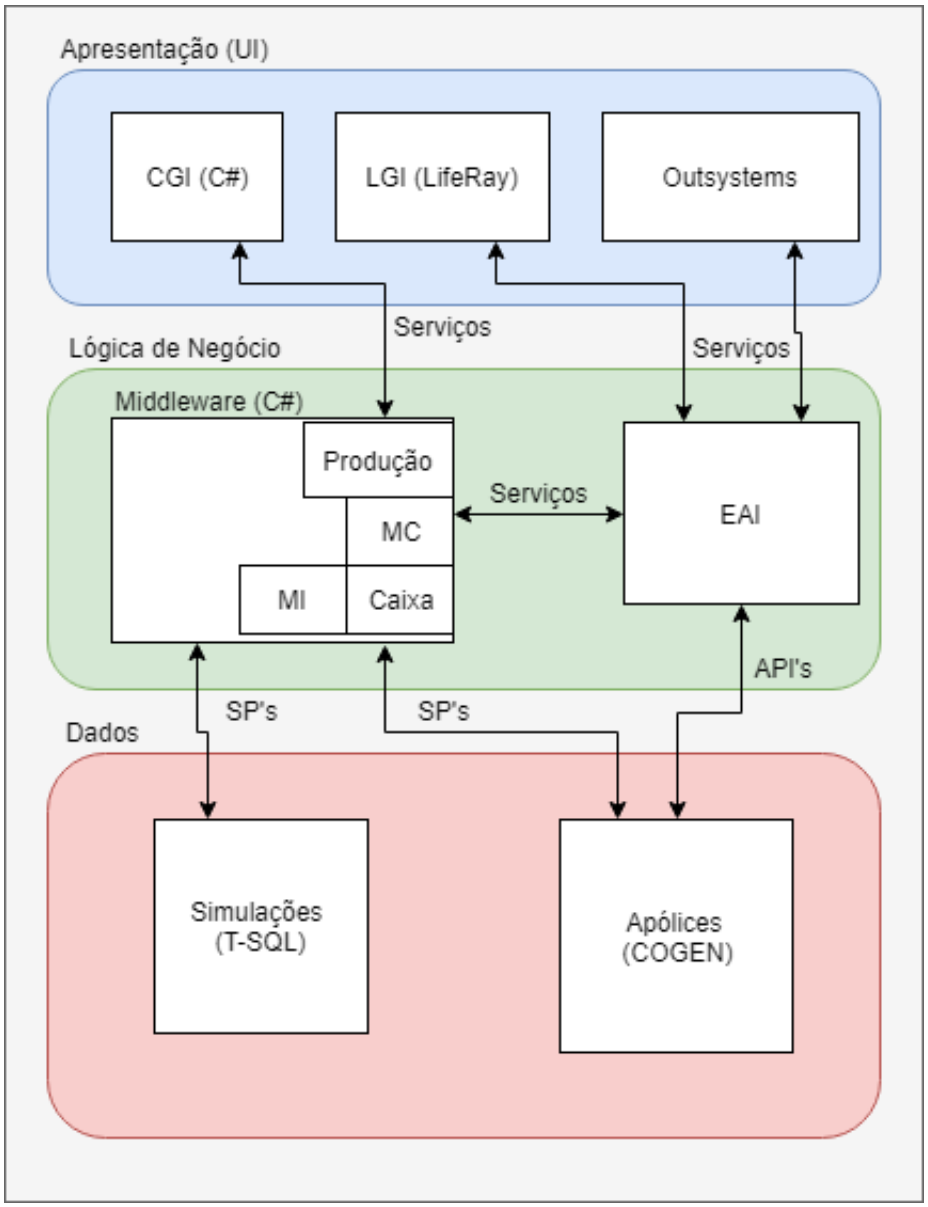

Figura 3.1: Arquitetura do Sistema
(MC- Motor de Cálculo)

3.2 Modelos para a geração de PDFs

Para a criação dos PDFs através do AOD, a companhia tem uma equipa que cria modelos para que depois os PDFs possam ser gerados de forma automática. Esta equipa tem por base o XML fornecido por *input* à ferramenta (neste caso o fornecido pelo Motor de Impressão), criando *forms* no modelo e utilizando para isso as *tags* do XML. Assim, quando o PDF for gerado, os valores provenientes do XML geram o documento com os dados em falta.

Desta forma, quando o MI chama o serviço para obter o PDF apenas tem de fornecer o id do modelo e o ficheiro XML com as informações do documentos sendo este preenchido de forma automática.

É assim criado um modelo para cada um dos documentos existentes na empresa. A cada um destes modelos é atribuído um id único, sendo a numeração destes agrupada por tipo de documentos (ex: os financeiros encontram-se entre o id 300 e 320).

3.3 O que é o Motor de Impressão?

O MI é responsável pela obtenção e expedição de documentos institucionais da companhia. A sua principal função é dar resposta a pedidos para a geração ou obtenção de um ou mais documentos, podendo enviá-los para destinatários específicos, por email, fax ou carta. Os documentos estão classificados em vários tipos e a cada um deles está associado um modelo (template) com a formatação e o conteúdo estático, presentes em todos os documentos desse tipo, e com *forms* a serem preenchidos com o conteúdo específico e dinâmico.

Assim, o MI tem apenas que, para cada documento pedido, obter o modelo associado, recolher a informação específica e enviar tudo para a ferramenta de geração de documentos, a qual irá tratar de ”preencher”o template com a informação específica, gerando um PDF. No final, o MI utiliza outro componente de forma a agrupar todos os PDFs num ficheiro único, sendo posteriormente expedidos para o destino desejado.

Na verdade, há casos em que o trabalho do MI se encontra facilitado, a informação específica vem no próprio pedido, não sendo necessário obter essa informação das bases de dados. Caso não venha por *input*, é o MI que recolhe essa informação, tendo uma lógica específica para cada documento ou grupo de documentos que se pretenda obter. Por outro lado, os documentos podem já terem sido gerados no passado e assim, há pedidos de documentos que não obrigam a utilizar os serviços da ferramenta de geração, mas o pedido pode explicitar se é ou não desejado um novo documento, obrigando a que seja gerada uma nova versão, mesmo quando o documento já existe.

Após o MI obter os dados específicos, estes podem ainda sofrer alterações ou ser necessário a adição de novas *tags* ao seu XML. Existem ajustes a dados (como o símbolo de moeda a utilizar, formato da data, etc) que se encontram guardados nas bases de dados

pois, podem não ser sempre constantes sendo por isso aplicados apenas após a obtenção do XML. Pode ainda existir a necessidade de aplicar regras para a adição de *tags* ao XML que correspondem a texto colocado pela ferramenta de geração (por exemplo o caso da proteção de dados, apenas é necessária em certos documentos com dados mais críticos e pode variar entre documentos). Estas regras encontram-se especificadas nas bases de dados do Motor de Impressão.

Por fim, o pedido ao MI contém o meio pela qual é pretendido que sejam expedidos os documentos. Deste modo, após conter o PDF agregado de todos os documentos, o MI faz o seu envio. No caso do envio ser feito por email, existe ainda uma base de dados que contém os *templates* genéricos de emails a acompanhar os documentos. Os meios de expedição podem ainda conter cartas que acompanham os documentos de forma a enquadrá-los, sendo estes também obtidos e expedidos pelo MI. Esta expedição é despoletada através de uma regra presente no *input* que define os documentos associados à expedição e no caso do eamil qual o *template* a utilizar como assunto e corpo do email bem como endereço de email de origem.

A utilização do MI é feita através de serviços que este expõe. Estes podem posteriormente ser consumidos diretamente por componentes que partilhem a mesma tecnologia (Microsoft) ou adaptados para serviços que possam ser consumidos por componentes de outras tecnologias.

3.3.1 *Input*

Como já foi referido anteriormente, a comunicação entre componentes do sistema é baseada na arquitetura SOA que utiliza serviços. De forma a uniformizar esta comunicação é utilizado como padrão a linguagem XML. O Motor de Impressão está preparado para que lhe seja pedido um conjunto de um ou mais documentos.

```
<Input>
    <NomePacote/>
    <CodTipoExpedicao/>
    <LstImprimir>
        <Imprimir>
            <Codigo/>
            <Grupo/>
            <ObterDados/>
            <DadosInput/>
            <IndVerificarDocExiste/>
            <ListaIdentificador>
                ...
            </ListaIdentificador>
        </Imprimir>
    </LstImprimir>
</Input>
```

Figura 3.2: Representação XML do *Input* do MI

Na figura 3.2 podemos ver como está definida a estrutura XML para o *input* do Motor de Impressão. O MI recebe como *input* a regra (NomePacote) e o meio de expedição (CodTipoExpedição), sendo que neste segundo caso temos carta, fax, email ou imprimir.

Na estrutura LstImprimir estão especificados um a um os documentos a obter em que o Código e Grupo definem os documentos para assim ser possível obter o modelo e, quando os dados não vêm por *input*, o Objeto de Dados ODs (classe C# com lógica de negócio necessária à obtenção dos dados específicos de um documento) a que esse documento está ligado, bem como a chave de tradução da lista de identificadores que é refletida no *input* dos ODs.

Relativamente aos dados, existem casos em que a informação específica de um documento é introduzida pelo utilizador, vindo neste caso as variáveis ObterDados e DadosInput abastecidas, conforme vísivel na figura 3.3.

```
<Input>
    <NomePacote>DocumentosCaixa</NomePacote>
    <CodTipoExpedicao Desc="">4</CodTipoExpedicao>
    <LstImprimir>
      <Imprimir>
        <Codigo>[illegible]</Codigo>
        <Grupo>[illegible]</Grupo>
        <ObterDados>false</ObterDados>
        <DadosInput>
            <DadosNotaDebCred>
                <DocumentoParaCaixa>S</DocumentoParaCaixa>
                <DtEmissao>2019-04-09</DtEmissao>
                <LocalEmissao>LISBOA</LocalEmissao>
                <NomeMorada>CENTRO DE TESTES SA</NomeMorada>
                <DebitoCredito>C</DebitoCredito>
                <NrMediador>99999999</NrMediador>
                <Morada1>RUA DE TESTES</Morada1>
                <Morada2>LISBOA</Morada2>
                <Morada3>1050-011 LISBOA</Morada3>
                <TipoAtipico>Empréstimo</TipoAtipico>
                <LstDetalheValor>
                    <DetalheValor>
                        <Detalhe>Empréstimo</Detalhe>
                        <Valor>50.00</Valor>
                    </DetalheValor>
                    <DetalheValor>
                        <Detalhe>ghdhfgbx</Detalhe>
                    </DetalheValor>
                </LstDetalheValor>
                <Extenso> São: Cinquenta Euros  </Extenso>
                <Total>50.00</Total>
            </DadosNotaDebCred>
        </DadosInput>
        <ListaIdentificador />
      </Imprimir>
    </LstImprimir>
</Input>
```

Figura 3.3: Representação XML do *Input* do MI para obtenção de uma nota de crédito

Em caso contrário, recorre-se aos Objetos de Dados (OD). Desta forma, juntamente com o a informação do Código e do Grupo, vem preenchida a lista de indentificadores

com elementos do tipo par chave-valor que servem para que seja criado o *input* do OD. Esta lista de identificadores encontra-se exemplificada na figura 3.4. Sendo a descrição da *tag* a variável do objeto de dados e o valor, o conteúdo dessa variável. Por exemplo, a variável "NrDeclaração" terá o valor "0000001009".

```
<Input>
    <NomePacote>ConsultaApolice</NomePacote>
    <CodTipoExpedicao Desc="Imprimir">4</CodTipoExpedicao>
    <LstImprimir>
        <Imprimir>
            <Codigo>[redacted]</Codigo>
            <Grupo>[redacted]</Grupo>
            <ObterDados>true</ObterDados>
            <IndVerificarDocExiste>true</IndVerificarDocExiste>
            <ListaIdentificador>
                <Identificador NomeCampo="NrDeclaracao">0000001009</Identificador>
                <Identificador NomeCampo="NrContrato">0200015353</Identificador>
            </ListaIdentificador>
        </Imprimir>
    </LstImprimir>
</Input>
```

Figura 3.4: Exemplo de caso em que lista de identificadores se encontra preenchida

3.3.2 Obtenção de dados e geração do documento

Numa fase inicial, cada documento a gerar é tratado de forma independente, sendo os documentos agrupados num único PDF no final. Desta forma, apenas é necessário explicar o fluxo de obtenção de cada documento e esse mesmo repetir-se-á para todos os documentos a obter.

Cada tipo de documento contém um código associado, existindo também o Grupo que define em que parte da aplicação este pode ser impresso (em diferentes partes da aplicação são utilizados diferentes "Grupos"pois existem documentos que é apenas permitida a sua impressão em certos ecrãs da aplicação, designados como pertencentes a "Grupos"diferentes). Este par código-grupo é utilizado para aceder à base de dados de forma a perceber qual a ferramenta utilizada para obter o documento (lógica já existente de migrações anteriores), qual o id do modelo que lhe corresponde e qual o objeto de dados responsável por recolher e agrupar toda a informação que é variável no documento.

De seguida, é necessário recolher os dados para fornecer ao AOD. Para isso, o MI tem no seu input duas variáveis (ObterDados e DadosInput). Caso a variável ObterDados vier com o valor "false"(falso), estes são obtidos através da variável DadosInput. Na maioria dos documentos de caixa (pagamentos), como é o caso das notas de crédito como a que tem o *input* representado na figura 3.3, os dados necessários para gerar o documento são todos recolhidos através de ecrã, deixando de estar essa lógica no lado do MI. Este XML não é o final que vai para a ferramenta de geração de documentos pois ainda pode ter de sofrer mapeamentos ou ser alvo de regras de inserção de novas *tags*. Caso isto não se verifique são utilizados Objetos de Dados (OD).

Os Objetos de Dados (ODs) são classes C# utilizadas para a obtenção da informação específica à geração de um documento. De forma a que não esteja toda a informação

ID_ObjectoDados	NrSeq	SourceName	TargetName
X	1	NrDeclaracao	NrDeclaraco
X	2	NrContrato	NrContrato

Tabela 3.1: Correspondência entre *input* do MI e de um OD

acomulada na mesma classe, o *input* e *output* dos ODs são objetos que este utiliza e encontram-se definidos na sua própria classe, permitindo assim que validações específicas do *input* sejam feitas na sua própria classe e levando a uma maior organização da estrutura de objetos.

A ListaIdentificador do *input* do MI é traduzida neste ponto para o *input* do OD. Desta forma existe uma tabela da base de dados que guarda a forma como esses campos devem ser traduzidos da ListaIdentificador para o OD. Por exemplo, no caso da figura 3.5 os campos NrDeclaracao e NrContrato são traduzidos através dos dados da base de dados conforme representado na tabela 3.1 sendo o SourceName o descritivo do Identificador e o TargetName o campo para o qual deve ser passado o valor.

```
<Input>
    <NomePacote>ConsultaApolice</NomePacote>
    <CodTipoExpedicao Desc="Imprimir">4</CodTipoExpedicao>
    <LstImprimir>
        <Imprimir>
            <Codigo>[illegible]</Codigo>
            <Grupo>[illegible]</Grupo>
            <ObterDados>true</ObterDados>
            <IndVerificarDocExiste>true</IndVerificarDocExiste>
            <ListaIdentificador>
              <Identificador NomeCampo="NrDeclaracao">0000001009</Identificador>
              <Identificador NomeCampo="NrContrato">0200015353</Identificador>
            </ListaIdentificador>
        </Imprimir>
    </LstImprimir>
</Input>
```

```
<Input>
    <NrDeclaracao>0000001009</NrDeclaracao>
    <NrContrato>0200015353</NrContrato>
</Input>
```

Figura 3.5: Representação XML do *Input* do MI vs Representação XML do *Input* de um OD

Os Objetos de Dados estão definidos segundo a estrutura orientada a objetos da linguagem C#. Logo, para documentos que possam ter várias variantes é criada uma classe principal que contém a informação que é comum a todos. De seguida, é criada uma classe para cada variante contendo a sua informação especifica. Por exemplo no caso das condições particulares existe uma classe ImprimirCondicoesParticularesBase que contém a informação geral a todos os documentos de condições particulares sendo depois criada uma classe especifica de cada Grupo de Produtos (Acidentes de Trabalho, Vida, Automóvel, etc). Nesta já são feitas validações consoante o produto, sendo a estrutura de input e output a mesma para todos os produtos do grupo.

A informação recolhida pelos objetos de dados é obtida através de *Stored Procedures* (SPs), sendo assim recolhida diretamente das bases de dados ou através de serviços, em que a informação é fornecida pela camada Core de negócio. Por sua vez os ODs tratam

e agrupam essa informação para estar assim organizada e "limpa" para que na ferramenta de geração de documentos seja feito apenas o mapeamento para o PDF.

O *output* do objeto de dados, representado na figura 3.6), é então serializado para um ficheiro XML. As classes C# que compõem o *output* do OD são assim serializadas para as diferentes *tags* do XML, sendo a *tag* principal a classe "mãe" e as *tags* que a constituem, os objetos dessa classe. Essa correspondência é representada na tabela 3.2.

Classe C#	***Tag* XML**
CabecalhoCls.cs	Cabecalho
DadosTomadorCls.cs	DadosTomador
DadosOSAutoCls.cs	DadosObjecto
DadosOSAutoBaseCls.cs	Auto
DetalheOSAutoCls.cs	DetalheObjecto
TabOpcaoCoberturasCls.cs	TabOpcaoCoberturas

Tabela 3.2: Correspondência entre classes C# e *Tags* do XML

Neste ponto, o Motor de Impressão contém a informação necessária para ir ao arquivo e perceber se o documento em questão já se encontra arquivado ou necessita de ser gerada uma primeira via através da ferramenta de geração de outputs. Deste modo, o *input* do MI contém uma variável (IndVerificarDocExiste) que caso venha com valor "true" (verdadeiro), é verificado no arquivo se o documento já existe, e, caso exista é obtida a segunda via do documento desse mesmo arquivo. Caso o IndVerificarDocExiste contenha o valor "false" ou o documento não esteja no arquivo, é necessário gerar um novo através do AOD. A verificação no arquivo só é feita neste ponto, porque por exemplo, no caso das declarações de seguro, estas podem encontrar-se em diferentes estados (como gravada ou emitida), sendo este estado apenas conhecido após obtenção dos dados. Caso o estado seja alterado, existe a necessidade de gerar um novo documento.

Preparação do XML e interação com a ferramenta

De forma a ser preparado para a interação com a ferramenta de geração de documentos, o XML passa por dois processos, um de mapeamentos e outro de aplicação de regras.

Os mapeamentos encontram-se guardados nas tabelas da base de dados do MI, estando identificados pelo id do modelo e podendo ser de dois tipos. Os de formatação servem para formatar moedas, datas, etc, existindo devido ao facto do formato geral das datas ou a moeda a utilizar poder mudar e assim esta informação encontra-se guardada na base de dados, pelo que em caso de variação esta mudança não tem de ser feita manualmente em todos os campos. Por outro lado, existem ainda os mapeamentos de filtragem de *tags*. Por exemplo, Os documentos de cariz financeiro utilizam o mesmo objeto de dados, pois o serviço que o MI consome para obtenção dos seus dados é comum a todos. Desta forma, de modo a enviar para o AOD apenas as *tags* necessárias a um determinado documento

```
<Output>
    <DadosGerais>
        <NrDeclaracao>0000002619</NrDeclaracao>
        <CodProduto>[illegible]</CodProduto>
        <DtInicio>04/12/2018</DtInicio>
        <DtInicioContrato>04/12/2018</DtInicioContrato>
        <DtFimContrato>01/01/2019</DtFimContrato>
        ....
    </DadosGerais>
    <DadosTomador>
        <NrCliente>1100304394</NrCliente>
        <NomeCliente>Testes SA</NomeCliente>
        <NIF>504918052</NIF>
        <Enderecamento>
            <nomemorada>Testes SA</nomemorada>
            <morada1>EN ANTIGA ESTRADA NACIONAL 1</morada1>
            <morada2>CASAL DA AMIEIRA</morada2>
            <morada3>2440-487 BATALHA</morada3>
            <NumeroPorta />
            <codpais>POR</codpais>
            <NomeDestinatario>>Testes SA</NomeDestinatario>
            <CodigoPostalMorada>2440-487</CodigoPostalMorada>
        </Enderecamento>
    </DadosTomador>
    <DadosSeguro>
        <CodObjectoSeguro>[illegible]</CodObjectoSeguro>
        <Auto>
            ....
        </Auto>
        <RiscosSeguros>
            <LstCobertura>
                ...
            </LstCobertura>
        </RiscosSeguros>
        <Credor>
            ...
        </Credor>
    </DadosSeguro>
</Output>
```

Figura 3.6: Representação XML do *Output* de um OD

Id Modelo	Campo Objeto de Dados	*Tag* XML
X	DocFin/DtDocumento	XML/dataemissaorecibo
X	DocFin/NrContrato	XML/numeroapolice
X	DocFin/NrSegurado	XML/numerocliente
X	DocFin/NrContribuinte	XML/numerocontribuinte
X	DocFin/NrRecibo	XML/numerorecibo
Y	DocFin/NrRecibo	XML/numerorecibo

Tabela 3.3: Exemplo de correspondência entre campos do objeto de dados e *Tags* do XML em documentos financeiros

financeiro, estas são filtradas do objeto de dados obtendo-se assim o ficheiro XML final. A tabela 3.3 exemplifica a forma como esta filtragem é feita.

De seguida, o XML ainda pode necessitar de regras para adição de novas *tags*, estando estas também identificadas pelo Id do modelo. Algumas destas regras refletem apenas detalhes como pequenos textos a adicionar, sendo gerais a cada modelo, como por exemplo, com o novo regulamento da proteção de dados é necessário que as simulações venham com essa informação. Como o MI é abstrato a nível de produto, existem ainda algumas destas *tags* que podem ser colocadas no XML consoante o produto pois o texto pode ser diferente. Assim, são guardadas em bases de dados de forma a que seja mais fácil a sua alteração pois, por não dizerem respeito à simulação ou apólice, são consideradas como "extras".

Neste ponto, o Motor de Impressão tem toda a informação necessária para que o documento seja gerado na ferramenta de geração de *outputs*. É então utilizado um serviço de modo a obter o documento, recebendo esse o id do modelo e o ficheiro XML com a informação relativa ao documento e retornando o *url* contendo o documento gerado.

Todo este processo de obtenção de dados e geração do documento é realizado para cada um dos documentos a obter. No final é utilizado um serviço em que é dado o *url* de cada um dos documentos gerados e é devolvido um *url* final com todos agrupados no mesmo PDF.

3.3.3 Expedição de documentos

Para além de obter os documentos, o Motor de Impressão tem também a capacidade de preparar o seu envio da forma pretendida pelo utilizador, que pode ser Fax, Carta ou Email. O processo de expedição encontra-se representado na figura 3.9.

Nestes casos existem regras (NomePacote), vindo descrita no *input* do MI qual a regra a utilizar. Estas regras são definidas consoante o meio de expedição ou o contexto em que se encontra o utilizador (ex: consulta de apólices, fim da simulação de contrato, etc) na camada de interface.

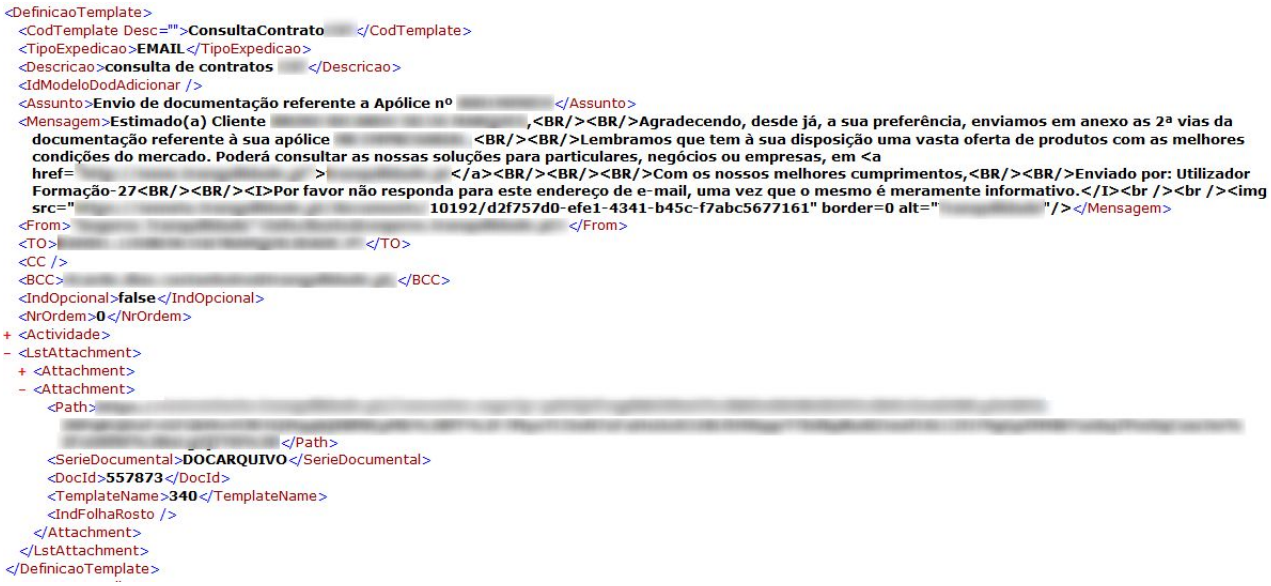

Figura 3.7: Exemplo de XML de expedição de email

Através desta regra, o MI consulta a base de dados de forma a recolher todas as informações necessárias à expedição e respetivo meio (indicado através da variável presente no *input* do MI CodTipoExpedicao). Por exemplo, no caso do envio ser feito por Email, esta é utilizada para obter o *template*, endereço de email de envio e assunto. Assim, os emails enviados são todos uniformes.

No caso do envio ser feito por Carta ou Fax, é necessário que um documento vá junto com os restantes documentos a enviar de forma a contextualizar o envio. Desta forma, através da regra é possível obter na base de dados quais os modelos que são dependentes à regra e ao meio de expedição e é seguido o fluxo de obtenção de documentos para assim obter esses modelos dependentes.

No final de termos toda a informação necessária ao envio dos documentos, é criado um pedido de expedição conforme exemplificado na figura 3.7. Consequentemente, a expedição é feita de forma assíncrona, chegando assim os documentos ao utilizador pelo meio desejado. Na figura 3.8 encontra-se o exemplo do email que chega ao utilizador.

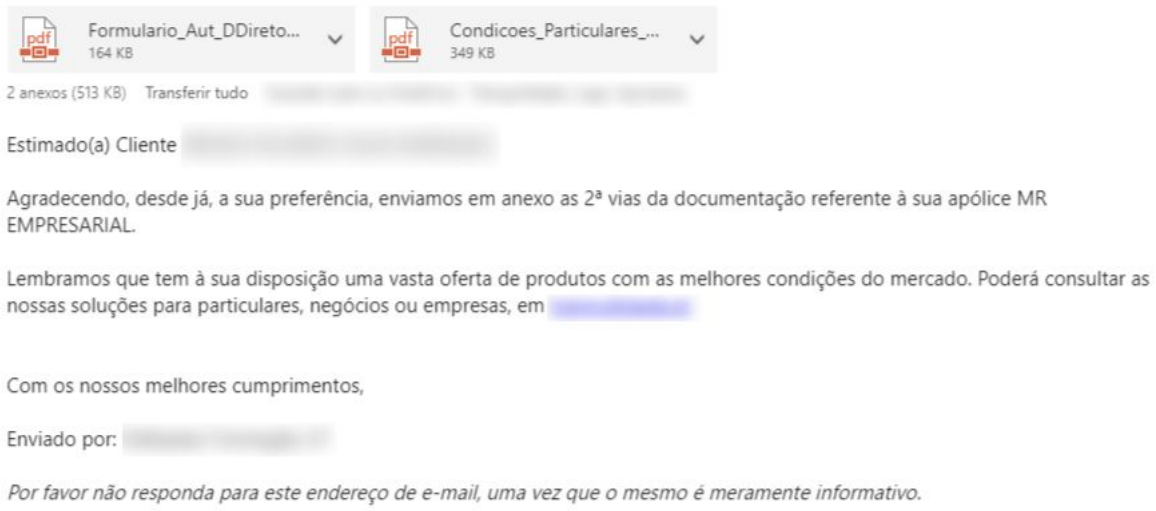

Figura 3.8: Exemplo de email

3.3.4 *Output*

O *output* do MI é composto por uma estrutura LstDocumento que contém um DocCls por cada um dos documentos obtidos. Este é preenchida com o número de sequência do respetivo documento, o *url* do PDF, o modelo e a descrição. Contém ainda a *tag* UrlDocUnico com o *url* do PDF com todos os documentos juntos num só ficheiro. Esta estrutura encontra-se representada na figura 3.10. Este *output* é utilizado caso o meio de expedição seja impressão, pois o resultado do *url* devolvido é mostrado no ecrã, sendo que nos outros meios de expedição serve apenas como referência para se saber que a expedição foi efetuada com sucesso.

```
<Output>
    <LstDocumento>
        <DocCls>
            <NrSequencia/>
            <Url/>
            <Modelo/>
            <DescricaoDocumento/>
        </DocCls>
    </LstDocumento>
    <UrlDocUnico/>
</Output>
```

Figura 3.10: *Output* do MI

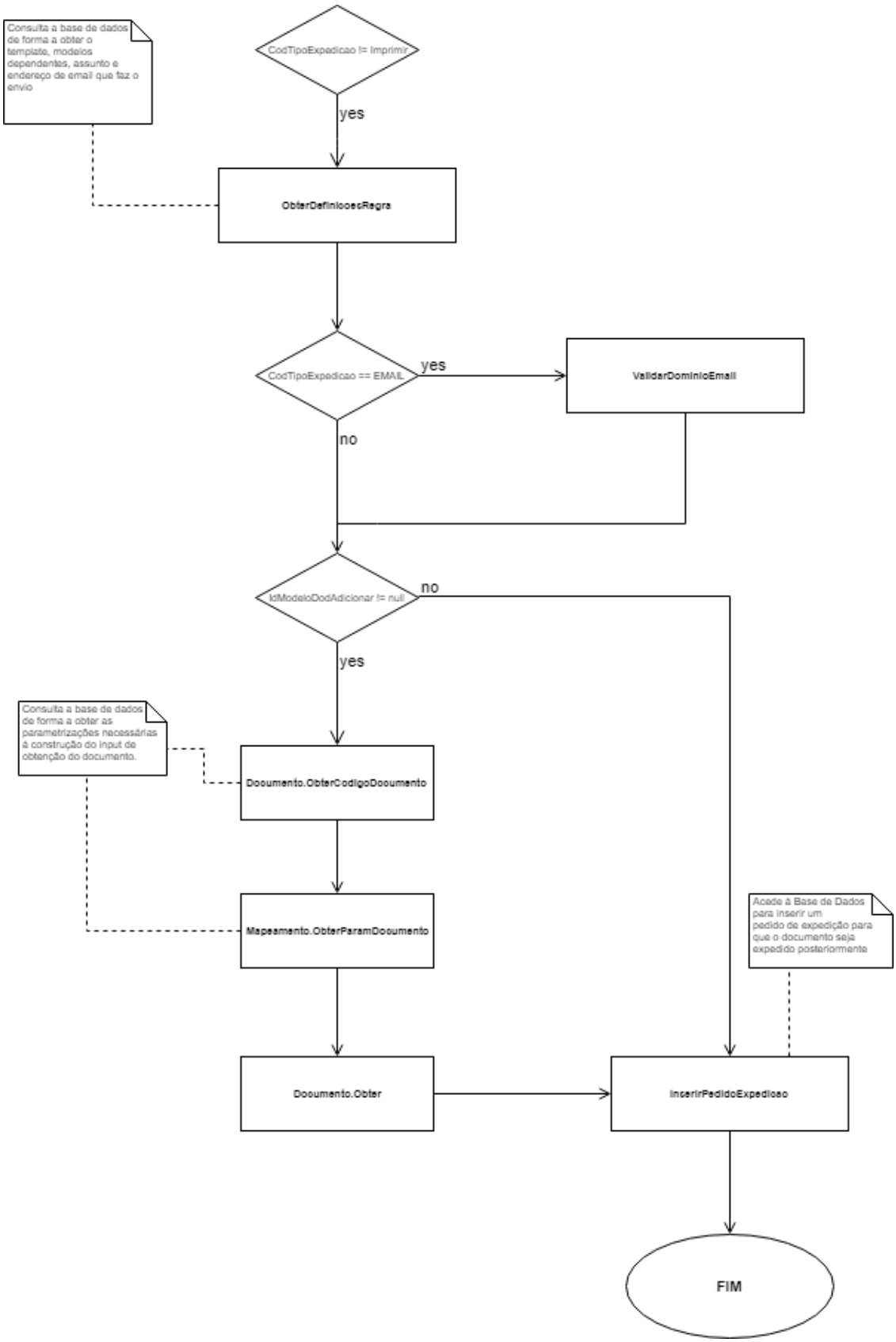

Figura 3.9: Fluxo de expedição de documentos

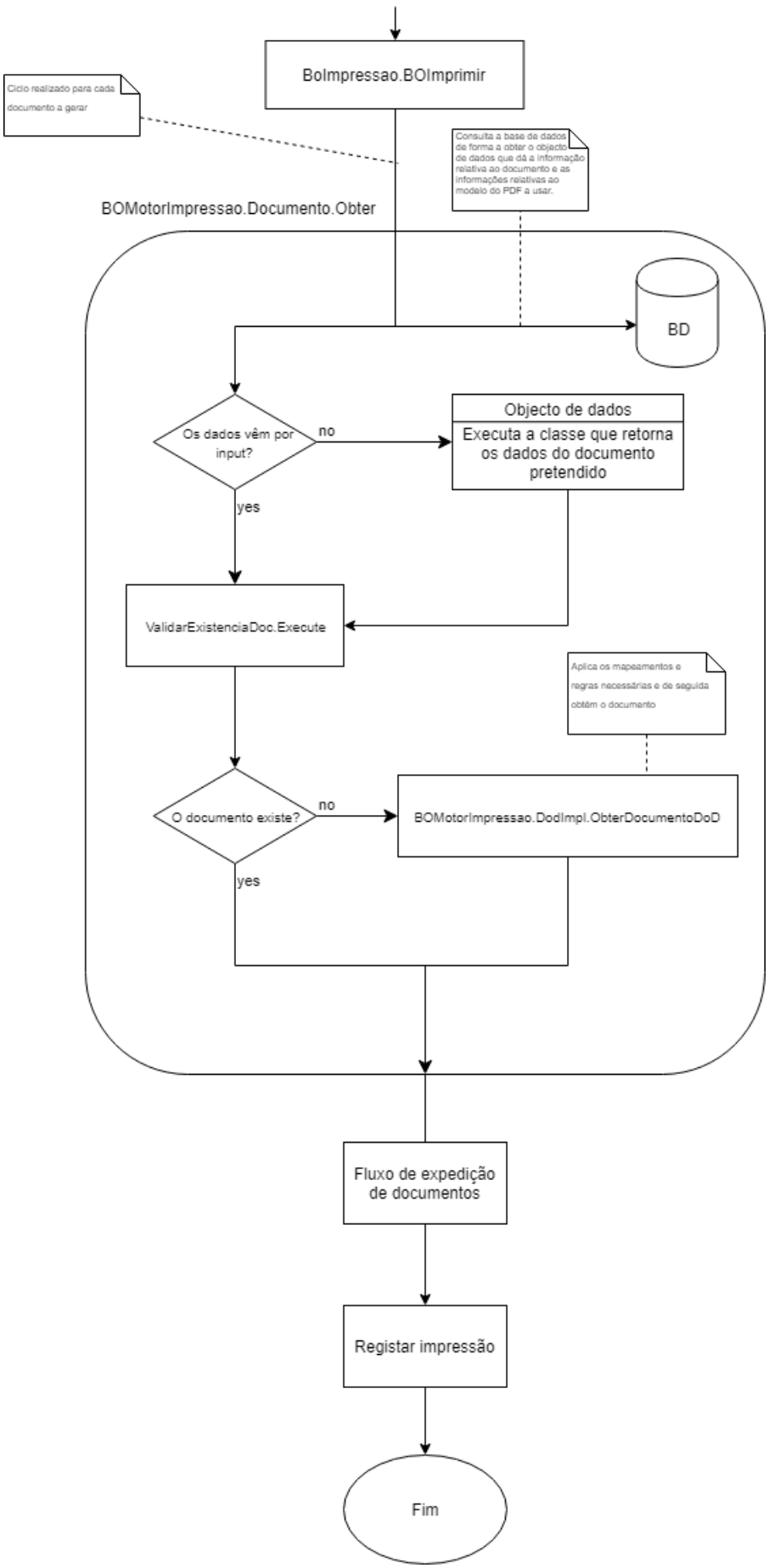

Figura 3.11: Fluxo do Motor de Impressão

3.4 Arquitetura do MI

O Motor de Impressão é composto por nove módulos (DLL), estando estes agrupados nos que se referem a serviços (os expostos e os consumidos pelo MI), os que contém toda a lógica inerente ao MI e os que definem a estrutura de *input*/*output* de cada uma das classes C# existentes na composição do MI. O esquema desta distribuição de módulos e a forma como interagem pode ser visível na figura 3.12.

- **FWKMotorImpressao:** expõe os serviços do MI, fazendo a sua ligação aos seus módulos, podendo estes serviços ser consumidos por outros componentes do sistema.
- **BOMotorImpressão:** contém a lógica relativa aos serviços expostos, sendo responsável por agrupar todas as peças do MI necessárias à obtenção do(s) documentos(s). O BOMotorImpressão interage com a ferramenta de geração de documentos e é neste que se insere o fluxo de envio de documentos, sendo por isso o seu *output* o *link* do PDF final.
- **MotorImpressao.WS:** é responsável pelo encapsulamento, através de *Wrappers* de serviços consumidos do componente de integração (EAI), de forma a colocar esses dados obtidos através de serviços na estrutura de classes desejada.
- **DoMotorImpressao:** recolhe das bases de dados, o id do modelo do documento a obter naquela iteração, a ferramenta de geração de PDFs e Objeto de dados a utilizar.
- **Prn.Dados**: contém os Objetos de Dados e é responsável por obter toda a informação, a nível de cliente e contrato, necessária à geração de um determinado documento. É aqui que é obtida a informação variável, ou seja, aquela que vai no ficheiro XML de *input* à ferramenta geração do documentos.
- **Types**: define as estruturas dos objetos utilizados nos diversos módulos.

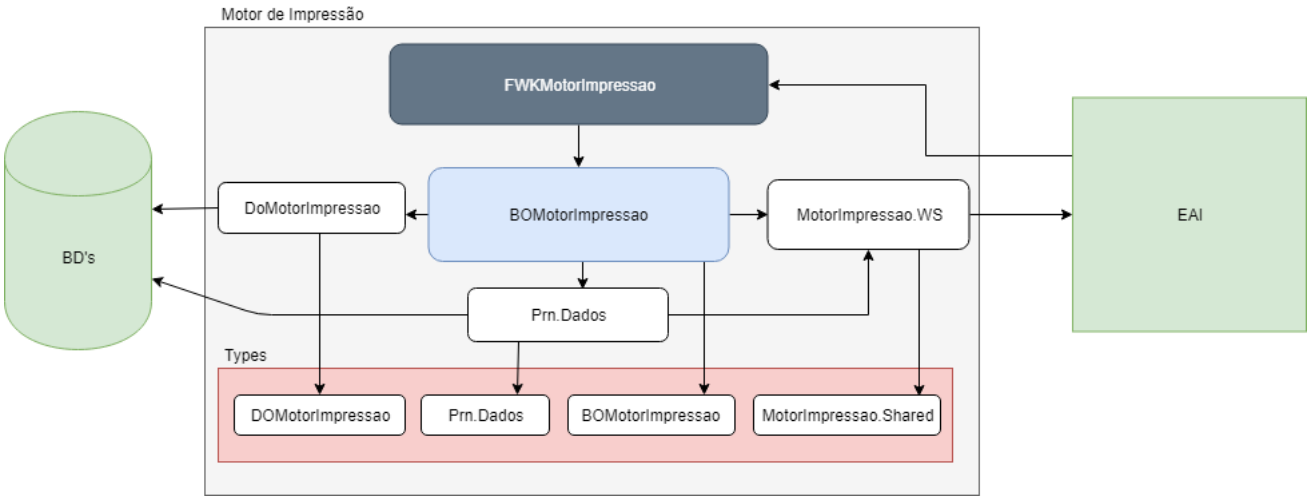

Figura 3.12: Arquitetura do Motor de impressão

3.5 Modelo de dados do MI

Como já foi referido anteriormente, o Motor de Impressão está otimizado de forma a ser abstrato, ou seja, estar preparado para imprimir qualquer tipo de documento para que, quando se quer adicionar um novo documento não ser necessário alterar o código que trata da geração de documentos mas sim apenas ter de adicionar lógica para ir buscar a informação relativa a esse documento. Desta forma, para todos os documentos gerados através do MI são guardadas as suas informações mais gerais de forma a que este tenha autonomia de as ir buscar, sendo para isso usada uma estrutura de *input*/*output* abstrata.

É por isso descrito neste capitulo a forma como a estrutura de tabelas do MI se encontra organizada e de que forma este possa usá-las e assim otimizar o seu desempenho.

As tabelas representadas na figura 3.13 têm assim, todas o seu papel na abstração da obtenção de documentos e de certos mapeamentos e regras aplicados.

- **Impressao_Documento:** É o local onde estão guardados o código e grupo que são únicos a um documento final a obter.

- **Impressao_Documento_ModeloDOD:** Liga os documentos ao seu modelo (id que corresponde ao *layout* na ferramenta que gera os PDFs). Desta forma é possível de forma abstrata associar por exemplo, um recibo a dois modelos diferentes dependendo do seu estado (se está ou não pago). É também possível que seja limitada a obtenção de documento documentos por canal (plataforma da qual são impressos), por moeda, companhia ou até mesmo meio de expedição.

- **Impressao_Modelo:** Aqui estão indicados o id do modelo e a ferramenta para que se possa definir a estrutura do XML e seus elementos.

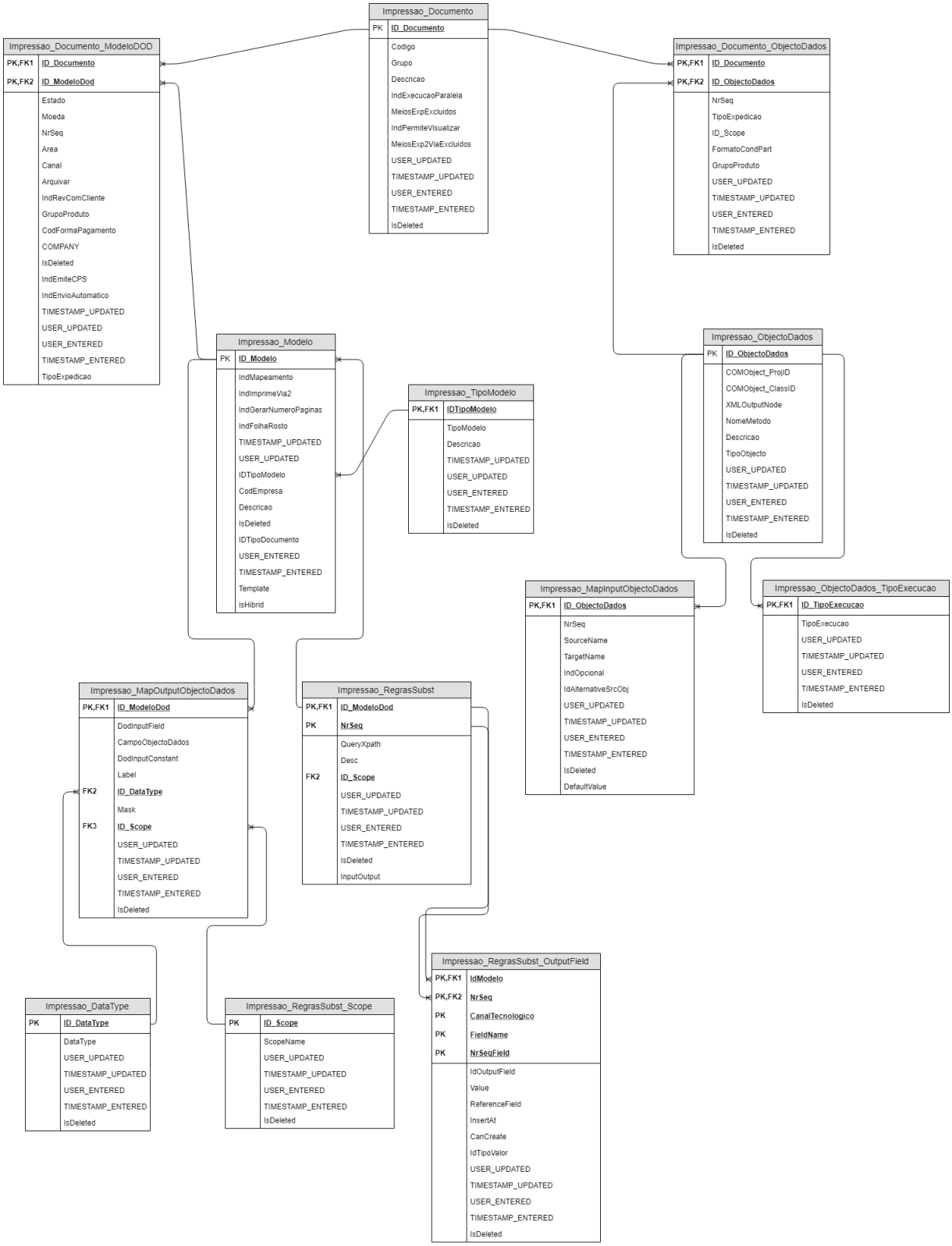

Figura 3.13: Modelo de Dados do Motor de Impressão

- **Impressao_MapOutputObjectoDados:** Serve para indicar possíveis mapeamentos que o XML de um documento possa ter. De forma a que para o mesmo Objeto de Dados sejam utilizados modelos diferentes, é possível aqui filtrar que *tags* devem ser retiradas consoante o modelo. Este também permite fazer formatações segundo a *framework* do sistema, como por exemplo a nível monetário, sendo muitas vezes apenas utilizado para este ultimo fim.

- **Impressao_RegrasSubst:** Parametriza regras a serem aplicadas no XML final antes de ir para a ferramenta que gera o PDF do documento. Estas regras podem ser aplicadas consoante o valor de determinada *tag* ou até mesmo adicionar ou mover *tags* no XML. Estas regras não são aplicadas dentro do Objeto de Dados mas sim com o recurso à base de dados pois são aplicadas apenas a um determinado produto de um modelo.

- **Impressao_RegrasSubst_Scope:** Existem regras ou mapeamentos que apenas se apliquem a um XML destinado a uma determinada ferramenta ou contexto. Deste modo, esta tabela define o âmbito em que a regra deve ser aplicada, ou seja, se a mesma se aplica ou não a uma ferramenta específica.

- **Impressao_Documento_ObjectoDados:** Esta tabela liga os documentos aos objetos de dados do qual é obtida a sua informação. Esta permite definir também qual o tipo de execução do objeto de dados (pré-processamento, processamento ou pós-processamento). Por vezes é necessário obter informações relativas aos dados chave de documentos antes ou depois destes serem processados, daí existir a necessidade de definir diversos tipos de processamento. É por isso possível que um documento esteja ligado a mais do que um objeto de dados.

- **Impressao_MapInputObjectoDados:** De forma a facilitar a chamada ao Motor de Impressão, generalizando a mesma, o MI é assim chamado com uma lista de indicadores que são convertidos no *input* do Objeto de Dados. Este está todo parametrizado com o nome que deve vir no *input* do MI e o nome desse mesmo campo no *input* do Objeto de Dados, bem como a indicação se este é ou não opcional.

- **Impressao_ObjectoDados:** Aqui são parametrizados todos os Objetos de Dados, a classe e módulo onde estes se encontram na solução da aplicação, bem como o método a utilizar para obter a informação que necessita.

A razão de existir esta parametrização toda deve-se ao facto de não só generalizar o Motor de Impressão a todos os documentos e colocar a parte específica nos Objetos de Dados, mas também no facto de certos pormenores não estarem de código de forma a facilitar a sua rápida alteração. Nalguns casos, como formatações de moeda por exemplo,

em caso de necessidade de alteração, é mais fácil efetuá-la na *framework* e assim, aplicar-se de forma automática a todos os documentos.

Desta forma, a inteligência passa para o próprio Motor de Impressão facilitando principalmente a forma de o adaptar a novos documentos ou suas alterações, pois a informação que necessita está nas bases de dados que o MI utiliza.

3.6 Os vários ambientes integrados da empresa

Na empresa onde o estágio foi realizado estão implementados quatro ambientes não produtivos (*staging*). Estes são réplicas do ambiente de produção que servem de suporte às várias fases de testes ao *software* desenvolvido. Apenas se coloca o *software* num ambiente caso este esteja validado no anterior.

Para este fim são utilizados os seguintes ambientes:

- *Work* (WK): É neste ambiente onde são primeiramente testadas as alterações efetuadas no sistema. Aqui é possível fazer *debug* dos desenvolvimentos sem se mudar o *software* para ambientes que estejam a ser utilizados por outros *developers* e de forma a tê-los logo disponíveis de forma integrada. Este é o primeiro ambiente onde se coloca o software.

- Desenvolvimento (DV): tal como o nome indica este é o ambiente onde são testadas as alterações feitas durante um desenvolvimento após serem validadas em ambiente de *Work*. Este é um ambiente mais estável, permitindo assim que se possa fazer testes de forma integrada sem que se perturbe os testes que estão a decorrer pelas respetivas equipas. O *software* só passa ao ambiente seguinte quando o mesmo estiver estável e possa entrar em fase de testes de aceitação.

- Testes de aceitação (TA): este é o ambiente onde se coloca *software* mais estável e aquele que é acedido pela equipa de testes. Serve para que as equipas de desenvolvimento possam disponibilizar o seu *software* quando este está pronto para que sejam iniciados os testes de qualidade sobre o mesmo. É neste ambiente que todo o *software* é testado e corrigido até estar pronto para entrar em ambiente de produção.

- Pré-Produção (PP): Este ambiente contém o *software* que está pronto para entrar em ambiente produção. É utilizado para testes integrados finais em ambiente muito parecido ao de produção de forma a testar o *software* num local mais limpo sem a possibilidade de ser perturbado por outros desenvolvimentos que ainda não terminaram a sua fase de testes.

- Produção (PR): Ambiente final, que é acedido e está disponível para o uso de todos os utilizadores.

Capítulo 4

Implementação

Neste capítulo é explicado o que foi necessário implementar no decorrer do projeto. Este implicou a passagem de 149 modelos de documentos da ferramenta AOD para o HPX. Devido à dimensão do projeto e à elevada quantidade de documentos a migrar, este dividiu-se em 6 fases (nomeadas de A a F). Em cada uma dessas fases foi migrado um conjunto de documentos, previamente selecionados, levando a que decorressem um conjunto de 6 iterações. No arranque de cada fase junto dos utilizadores, foi necessário realizar o respetivo acompanhamento, de modo a atenuar problemas que pudessem existir. O facto de o projeto ter sido faseado também contribuiu para a diminuição do impacto destes problemas e consequentemente facilitar essa monitorização.

Antes da realização deste projeto foi criada uma estrutura para o XML de forma a que esta contenha a mesma base (LstDocumento/Documento) e, para que, para certos modelos e quando necessário, passou a ser possível enviar o conteúdo relativo a vários documentos. Assim, começaram a ser gerados vários documentos num único PDF numa única execução da ferramenta. Esta estrutura é colocada no XML relativo ao modelo quando este se encontra associado à ferramenta HPX, tendo sido necessário efetuar esta associação nas bases de dados do MI.

Consequentemente, as regras e mapeamentos necessitaram de ser alterados. Estes têm por base o caminho (*path*) até à *tag* a alterar, filtrar ou até ao local onde esta deve ser adicionada. Assim, este caminho alterou-se com a nova estrutura e com o facto de poderem existir mais do que um documento num XML.

Com a mudança de ferramenta, também mudam as suas especificações. Deste modo, foi necessário fazer alterações nos Objetos de Dados, de forma a que certa informação fosse de acordo com as especificações do HPX. Existiram casos em que, de forma a aproveitar melhor a nova estrutura de lista de documentos, foi necessário que ODs fossem criados de novo e adaptados de forma a se adaptarem de uma forma eficiente. Por exemplo, no caso das Declarações de Seguro passou-se a ter uma Declaração por objeto assegurado pela apólice. Neste caso, todas as Declarações relativas a uma apólice começam a ir no mesmo ficheiro XML, diminuindo a interação com a ferramenta e evi-

tando a latência a nível de comunicação. Certas incoerências nos dados foram corrigidas de forma a melhorar a qualidade dos documentos ao serem migrados. Em casos que a informação recolhida pelos ODs vinha de outros componentes, foi necessário alterar a forma de consumir o serviço, para que a informação fosse recolhida de forma correta.

Da parte da equipa que cria os modelos do lado do HPX, foi necessário que fosse criado um novo modelo por cada documento a migrar. Estes modelos foram criados aproveitando as vantagens da nova ferramenta, existindo casos como a morada, em que foi criado um modelo base pois todos os documentos partilham os campos de morada. Para documentos semelhantes e estruturas diferentes, os modelos foram criados de forma hierárquica, ou seja, criada uma base para todos os modelos do mesmo tipo de documentos e depois modelos que estendem esse com a informação específica relativa ao produto. Neste ponto, foi necessária uma grande comunicação entre equipas pelo que se necessitou de disponibilizar o novo XML dos modelos de modo a que esta equipa pode-se criá-los com base nesse XML.

Por fim, cada documento foi testado de forma unitária e integrada de modo a garantir que as alterações foram bem efetuadas e obtido o resultado esperado. Ainda foi criada, tanto documentação relativa às alterações realizadas no decorrer do projeto, bem como da forma de imprimir de cada tipo de documento e como obter dados para essa impressão nas bases de dados.

4.1 Nova estrutura do XML

Antes deste projeto, foi criada uma nova estrutura (LstDocumento/Documento) para o ficheiro de XML a enviar para a ferramenta de geração de PDFs. Com a migração para o HPX, esta estrutura passa a ser utilizada, permitindo assim que sejam, enviados na mesma utilização da ferramenta, vários documentos desde que pertençam ao mesmo modelo. Um exemplo desta nova estrutura encontra-se representado na figura 4.1.

Para que esta alteração se verifique em cada modelo, é necessário que a sua associação à ferramenta seja alterada na base de dados em que é definida. Deste modo, os documentos que adquiriram esta nova estrutura foram associados a esta nova ferramenta na tabela Impressap_Modelo. Esta alteração foi feita para cada documento a passar para a nova estrutura com a migração.

Na adoção desta nova estrutura, existiram Objetos de Dados que, de forma a se adaptarem à nova estrutura de lista de documentos e diminuírem o número de chamadas à ferramenta, tiveram de ser alterados ou até criados de novo devido ao tamanho das suas mudanças. Esta alteração reflete-se na existência de uma estrutura Documento por cada um dos documentos a gerar desse modelo. Um exemplo desta mudança são as Declarações de Seguro, sendo detalhado na secção 4.2.

Os documentos de cariz financeiro contém o mesmo Objeto de Dados, sendo a sua

```
<LstDocumento>
    <Documento>
        <Cabecalho>
            ...
        </Cabecalho>
        <TabInfoRodape>
            ...
        </TabInfoRodape>
        <DadosEnderecamento>
            ...
        </DadosEnderecamento>
        <DadosTomador>
            ...
        </DadosTomador>
        ...
    </Documento>
</LstDocumento>
```

Figura 4.1: Exemplo de estrutura LstDocumento

estrutura específica definida através de mapeamentos que filtram esse *output* de forma a recolher apenas as *tags* necessárias para esse modelo. Por este motivo, esta estrutura manteve-se com a migração, tendo sido criada uma nova variável isHibrid, na tabela Impressao_Modelo que indica que o modelo em questão já foi migrado mas mantém a mesma estrutura. A avaliação para saber se a estrutura do modelo era ou não mantida foi feita para cada um dos documentos a migrar.

Consequentemente, foi necessário alterar as regras de substituição relativas a cada modelo pois como a estrutura do XML é diferente, o local onde devem ser colocadas as *tags* também muda (por exemplo, tags que estavam na raiz deixa de o estar pois a estrutura LstDocumento é que passa a estar na raiz). Isto verifica-se também a nível de mapeamentos a aplicar às *tags* sendo adaptados ao novo formato XML pois o *path* a nível de XML muda e o campo a mapear agora pode aparecer mais que uma vez, sendo que o mapeamento deve estar pronto para isso.

4.2 Criação de um Objetos de Dados

No decorrer do projeto existiu um Objeto de Dados que necessitou de uma alteração muito extensa pois foram necessárias muitas *tags* alteradas com o decorrer da criação do novo modelo para o documento. Como consequência, foi necessário criar um novo Objeto de Dados para que esta alteração não colidisse com o que já existia em ambiente de produção relativo à ferramenta antiga.

Esta tarefa dividiu-se então em vários pontos:

1. Perceção da conclusão do novo serviço. Pertencendo este à mesma camada e sendo

a tecnologia a mesma, não foi necessária a criação nem alteração de nenhum *wrapper*.

2. Modificação do *input* de forma a adequá-lo à estrutura do novo serviço e do *output* do Objeto de Dados, passando este a conter a nova estrutura do documento. Para isto, foi necessário a criação de novas estruturas semelhantes à do *output* do serviço para que este pudesse ser consumido da melhor forma, visto a estrutura do original ter mudado de forma radical.

3. Criação de toda a lógica de negócio necessária para a geração do documento, ou seja, de como o Objeto de Dados preenche o seu *output* de forma a conter toda a informação requerida pelo cliente. Consequentemente, foi necessário ir buscar os dados e aplicar-lhes todas as regras de negócio de forma a estes virem formatados de acordo com os padrões da empresa.

4. Colocação da informação do novo Objeto de Dados na base de dados. Como foi explicado, necessitou-se de colocar a informação do novo Objeto de Dados na base de dados, bem como o *input* que este deve conter e, por fim, liga-lo ao par codigo-grupo do documento correspondente. Deste modo, quando este for pedido no Motor de Impressão com o código correspondente, será este o Objeto de Dados chamado.

5. Coordenação com a equipa que cria os modelos na ferramenta de geração de PDFs de forma a que o desenho esteja conforme o pedido. Consequentemente realizou-se alterações no Objeto de Dados, no seu *output* e informação nas bases de dados para assim estarem de acordo com o necessário a gerar o PDF.

Um exemplo de uma nova versão de Objeto de Dados criada foi o caso do que recolhe e trata a informação relativa ao documento de Declarações de Seguro.

Como o HPX permite gerar documentos do mesmo modelo para ficheiros XML diferentes, a estrutura de DadosSeguro das Declarações de Seguro passaram a ter uma estrutura Objeto Seguro para que esta possa ser geral a todos os produtos e não específica a um determinado produto. Esta nova estrutura pode ser visível na figura 4.2.

Ainda nesta mesma figura, verifica-se que a estrutura Documento aparece duas vezes. Isto deve-se a uma regra de negocio que diz que Objetos de Seguro diferentes devem vir em Declarações distintas. Desta forma, a nova estrutura permite que não sejam necessárias várias chamadas ao serviço de geração de documentos, sendo obtidos dois documentos numa só execução. Foi ainda desenvolvida a lógica de negócio para a recolha de dados de cada uma das estruturas presentes no *output* representado na figura.

```
<LstDocumento>
    <Documento>
    <Documento>
        <DadosGerais>
            <NrDeclaracao>1000001663</NrDeclaracao>
            <CodProduto>[illegible]</CodProduto>
            <DtInicio>30/01/2019</DtInicio>
            <DtInicioContrato>30/01/2019</DtInicioContrato>
            <DtFimContrato>31/01/2019</DtFimContrato>
            ...
        </DadosGerais>
        <DadosTomador>
            <NrCliente>4120459634</NrCliente>
            <NomeCliente>Testes SA</NomeCliente>
            <NIF>504918052</NIF>
            <Enderecamento>
                <nomemorada>Testes SA</nomemorada>
                <morada1>EN ANTIGA ESTRADA NACIONAL 1</morada1>
                <morada2>CASAL DA AMIEIRA</morada2>
                <morada3>2440-487 BATALHA</morada3>
                <NumeroPorta />
                <codpais>POR</codpais>
                <NomeDestinatario>>Testes SA</NomeDestinatario>
                <CodigoPostalMorada>2440-487</CodigoPostalMorada>
            </Enderecamento>
        </DadosTomador>
        <DadosSeguro>
            <ObjectoSeguro>
                <CodObjectoSeguro>[illegible]</CodObjectoSeguro>
                <CodEstatistico>[illegible]</CodEstatistico>
                <Auto>
                    ...
                </Auto>
                <LstCobertura>
                    ....
                </LstCobertura>
                <Credores>
                    <LstCredor>
                        <CredorDeclCls>
                            ...
                        </CredorDeclCls>
                    </LstCredor>
                </Credores>
            </ObjectoSeguro>
        </DadosSeguro>
    </Documento>
</LstDocumento>
```

Figura 4.2: Representação XML do *Output* alterado das Declarações de Seguros

4.3 Alteração de Objetos de Dados

Anteriormente a este projeto, foi criada uma interface e por consequente uma classe abstrata que a implementa designada de ContextoImpressão. Esta classe é assim estendida pelos *inputs* das classes que necessitem dessa informação de forma a que os Objetos de Dados pudessem estar contextualizados do código e grupo do documento a obter, bem como da ferramenta a ser utilizada. Esta informação é preenchida pela *framework* no momento de criação do input dos ODs. Desta forma, foi possível distinguir as alterações por ferramentas e que estas pudessem coexistir em tempo de migração.

```
newPS.IndSelecionada = (input.ContextoImpressao != null &&
    (input.ContextoImpressao.TipoModelo ==  Enums.ETipoModelo.HPX || input.ContextoImpressao.IsHybrid.BooleanValue)) ?
        Constants.IMPRESSAO.CHECKBOX_CHAR_FALSE :  Constants.BOOLEAN.FALSE;
```

Figura 4.3: Exemplo de código com distinção de ferramentas

Na figura 4.3 é possível ver um excerto de código em que é feita esta mesma distinção entre ferramentas. Esta imagem demonstra também uma das alterações do AOD para o HPX que foi o conteúdo das *checkboxs*. Estas passaram a ter um caracter que as representa o facto de ser positiva ou negativa.

Outro caso foi a remoção de *tags* em excesso, pois no AOD era necessário colocar, por exemplo uma *tag* com a indicação que não existiam coberturas para que a tabela das coberturas não fosse apresentada. No HPX essas *tags* foram retiradas pois são informação "a mais" e esta ferramenta já não mostra a tabela com a ausência da *tag* LstCoberturas, não necessitando de qualquer extra.

Uma das vantagens do HPX é, para tabelas, a informação é preenchida na tabela mais à direita, sendo possível a omissão de colunas em caso de inexistência de informação.

AOD	Col1	Col2	Col3	Col4
Linha1	X	Y	W	Z
Linha2	X1		W1	Z1
Linha3	X2	Y2	W2	Z2

Tabela 4.1: Exemplo de tabela em AOD

HPX	Col1	Col2	Col3	Col4
Linha1	X	Y	W	Z
Linha2	X1	-	W1	Z1
Linha3	X2	Y2	W2	Z2

Tabela 4.2: Exemplo de tabela em HPX

Essa característica do HPX leva a que seja necessário adicionar um hífen no local de uma variável vazia para que a informação não fique trocada entre colunas. Assim, são observadas as diferenças representas nas tabelas 4.1 e 4.2.

4.4 Alteração na forma de consumir um serviço

Como o sistema está assente em várias tecnologias, é necessário que sejam utilizados serviços de forma a garantir a compatibilidade e abstração na comunicação entre componentes baseados em diferentes tecnologias. O MI tem assim de consumir serviços de forma a obter informações relativas a dados que se encontrem em BDs que não são geridas pelo componente *Middleware* (MW).

Devido a problemas detetados em documentos durante o processo de migração, foi necessário alterar os dados que compunham o ficheiro XML. Como estes dados eram obtidos a partir de outro componente, foi necessária a alteração de serviços e posteriormente alterada a forma como estes são consumidos do lado do MW. Existiu, portanto, o pedido de alteração do serviço, bem como a sua estrutura.

Primeiramente, foi necessário alterar a documentação relativa ao serviço para conter os novos campos a serem consumidos pelo MI. Esta documentação, além de manter atualizada a informação da estrutura e funcionamento do serviço, permite fazer o pedido à equipa de que gere o componente de integração (EAI). Isto permite ao MW ter controlo sobre a forma como os dados chegam e de como deve estar preparado para os consumir.

Após o serviço ser alterado, este foi especificado através da linguagem WSDL. Uma das vantagens do C# é que este converte a especificação do serviço numa estrutura de classes, facilitando assim o desenvolvimento.

Consequentemente, necessitou-se de alterar o *wrapper* que faz a "tradução" dos dados do serviço para o objeto de dados, permitindo assim fazer um tratamento dos dados de forma a que estes cheguem de uma forma específica.

Dentro dos *wrappers* da solução, estes dividem-se em dois tipos: os genéricos da qual a sua alteração ou adição de novos campos não altera a forma como estes se apresentam em produção e os mais específicos em que a sua alteração pode afetar outros ambientes quando a mesma ainda não foi efetuada no HPX. Este segundo caso pode afetar a forma como estes dados são apresentados no PDF final. Consequentemente, é necessário o uso de *feature* (variável booleana guardada na base de dados que permite que seja seguido o fluxo antigo em caso de problema ou falta de dependências de outros componentes) de modo a ser mais fácil o controlo na disponibilização das novas funcionalidades ou alterações.

Foram de seguida feitas as modificações necessárias ao Objeto de Dados e ao *output* deste para que a informação retornada esteja adaptada à nova estrutura do documento e à forma como a ferramenta está preparada para consumir o que é devolvido pelo MI.

4.5 Documentação

Na equipa são adotados vários tipos de documentação. Quando é alterado ou criado algum método C# novo, este deve ser documentado segundo a documentação base e

templates existentes.

No decorrer do projeto esta baseou-se essencialmente em dois pontos: criação/alteração de métodos e documentação própria ao estado do projeto. No primeiro caso, é seguido o modelo apresentado na figura 4.4. Primeiro deve ser feita uma descrição do que é o método. Caso este seja ou utilize um *webservice*, deve ser identificado para ficar associado. Deve conter ainda uma representação XML da sua estrutura de *input*/*output* do método. De seguida deve ser descrita a interface, ou seja, indicar o tipo de cada atributo de *input*/*output* e fazer uma pequena descrição (regras de validação, como é preenchido, etc). Devem ser depois detalhadas as ações principais do método e o comportamento esperado.

Figura 4.4: Estrutura da documentação

Relativamente à documentação do próprio projeto, foi criado um ficheiro Excel com os dados indicativos ao projeto. Neste ficheiro é indicado se o documento é ou não obtido pelo Motor de Impressão, as alterações que necessita e algumas observações percebidas ao longo da migração. É ainda neste ficheiro que se encontra detalhado o Documento, ou seja, o tipo da sua estrutura (se este é híbrido ou se é a do HPX) e como pode ser impresso.

No decorrer do projeto foi ainda criado um documento adicional com uma lista de modelos de documentos. Para cada um destes modelos foi identificado como estes podem ser impressos na aplicação web da companhia. No caso de documentos que sejam impressos

na consulta de contrato foram indicadas as condições que levam a que esse documento surja e no caso dos recibos as tabelas da base de dados e condições à sua impressão.

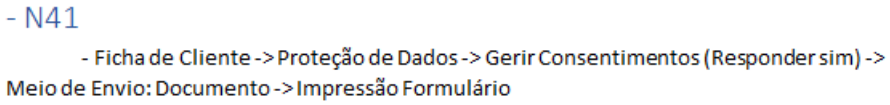

Figura 4.5: Exemplo de documentação de como imprimir um determinado documento

4.6 Testes Unitários e Integrados

Para cada um dos documentos do motor de impressão incluídos no projeto de migração, foi necessário que se efetuassem testes unitários e integrados de forma a melhorar a qualidade da entrega de cada um dos documentos.

Para a geração dos documentos era necessário perceber as condições em que estes eram impressos, e procurar nas bases de dados apólices em que estes poderiam ser obtidos, sendo por vezes necessário criar as próprias condições para a sua geração.

De seguida, eram guardadas as evidências dos documentos em formato PDF ou com o *log* da sua obtenção. Eram depois anotados erros oriundos de outras equipas para aviso posterior e corrigidos os encontrados relativos ao MI.

Área	Meio Expedição	XML	Prefixo	Id Mod	Variant	Descricao	Sessão TA
Produção	Batch e Online			G211		Carta Verde	MWJVHTE6BX64EA 10/05 15:29:35
Financeira	Batch e Online			G300		Aviso Multibanco	MWJVHTE6BX64EA 10/05 15:37:29
Financeira	Batch e Online			G304		Fatura Recibo	MWJVI84XB04NX6 10/05 16:19:41
Financeira	Batch e Online			G305		Aviso de Incobrança	MWJVI84XB04NX6 10/05 16:21:03

Figura 4.6: Exemplo de evidências de testes

Por fim era guardado o estado do documento e o ambiente até ao qual se encontrava o seu desenvolvimento.

Existem documentos que apenas são impressos em casos muito específicos sendo para isso necessários casos de testes muito particulares e de difícil criação. Para isso recorre-se regularmente às bases de dados para não termos de criar novos ambientes e assim, podermos testar as nossas alterações, otimizando o tempo despendido em testes unitários.

4.7 Acompanhamento após disponibilização junto dos utilizadores

Após cada grupo de migração ser disponibilizado aos utilizadores, foi necessário verificar se existiam erros na obtenção desses documentos. Estes erros ocorriam pois o

tamanho dos campos dos documentos gerados não correspondia aos definidos na criação do modelo. Neste caso, a identificação dos erros é mais fácil do lado do MI pois é despoletado um erro desse lado, daí ser feita no MI e aí era necessário informar a equipa que gere os modelos para proceder à correção.

Capítulo 5

Resultados

De forma a garantir a qualidade do projeto, este foi testado pela Equipa de Testes de qualidade da companhia. No final de cada uma das 6 fases de implementação da migração (A a F), o projeto passou assim por testes de qualidade e de aceitação, estando por isso os resultados também divididos em diferentes grupos, correspondendo a cada uma dessas fases de migração.

Estes modelos foram divididos consoante a sua complexidade e o grupo de documentos ao qual pertencem de forma a facilitar os desenvolvimentos e testes.

Como grande parte dos documentos gerados pelo Motor de Impressão são obtidos através da plataforma web da empresa, estes foram testados seguindo o fluxo normal de um utilizador e assim obter o documento. De seguida, foi comparado com a versão do documento obtido no AOD de forma a perceber se o seu conteúdo se mantinha correto após a migração. Sempre que se verificava falta ou incoerência de dados a Equipa de Testes consultava as equipas de desenvolvimento para perceber se o problema estava nos dados de teste ou necessitava de alteração.

Sendo o objetivo principal desta tese as alterações ao MI, os resultados apresentados em seguida refletem apenas os documentos obtidos através do MI, logo não são apresentados os resultados de documentos gerados por outra via.

5.1 Gerais

No decorrer deste projeto, foram migrados 149 documentos que são gerados via MI. Destes 149 documentos, 128 deles foram migrados com sucesso, sendo que dos restantes 7 já tinham sido migrados anteriormente e 14 foram considerados como descontinuados pela companhia.

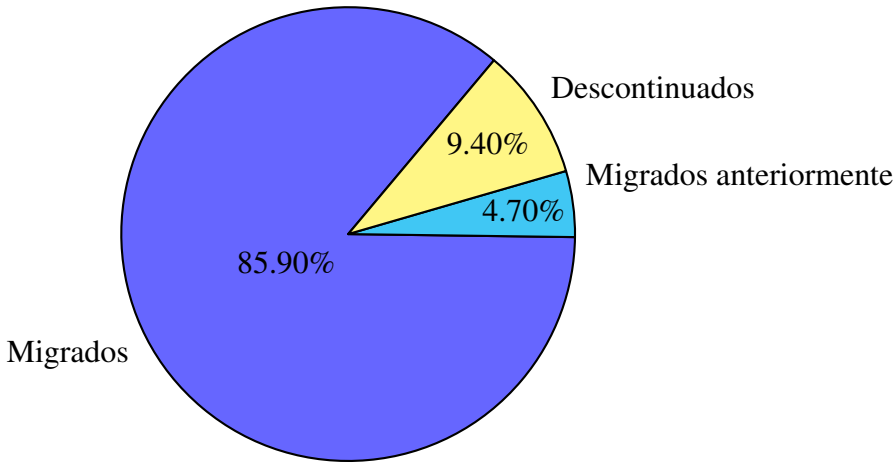

Figura 5.1: Resultados da migração

5.2 Grupo A

Neste grupo foi proposta a migração de 24 documentos gerados através do Motor de Impressão. Destes apenas 22 foram testados pois dois deles já tinham sido migrados anteriormente na sequência de alterações necessárias.

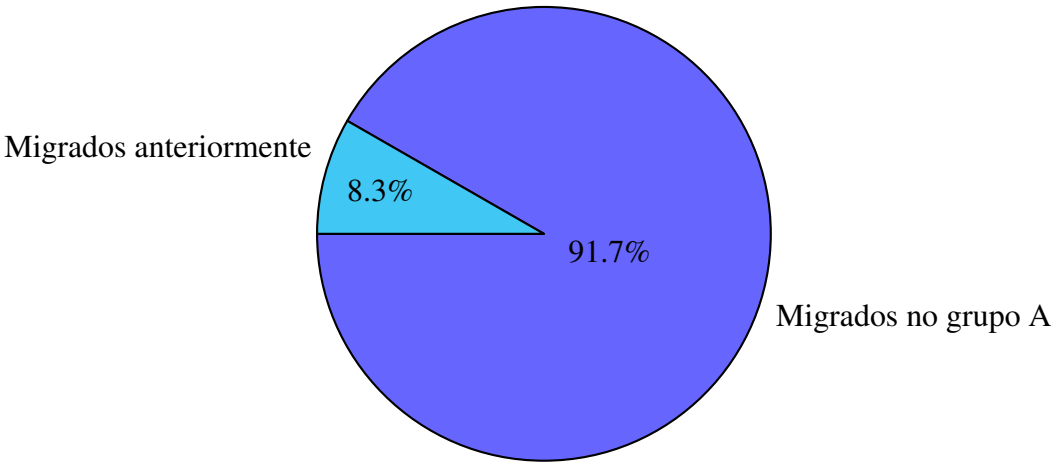

Figura 5.2: Resultados da migração do Grupo A

5.3 Grupo B

Dos 64 documentos que faziam parte deste grupo de migração, 56 deles foram migrados com sucesso, 4 deles já tinham sido migrados anteriormente, sendo que 4 foram descontinuados.

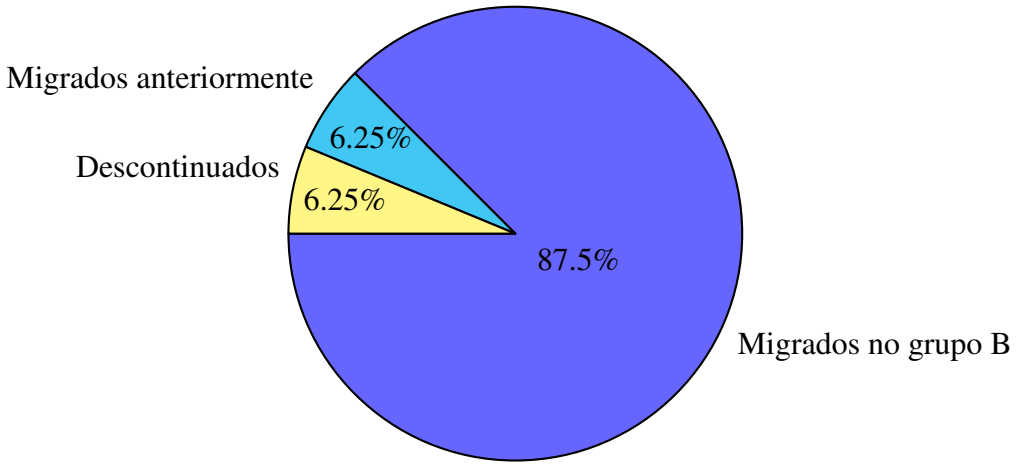

Figura 5.3: Resultados da migração do Grupo B

5.4 Grupo C

O Grupo C era composto por 16 documentos, dos quais 12 deles foram migrados com sucesso sendo que 4 deles foram descontinuados por desuso dos utilizadores.

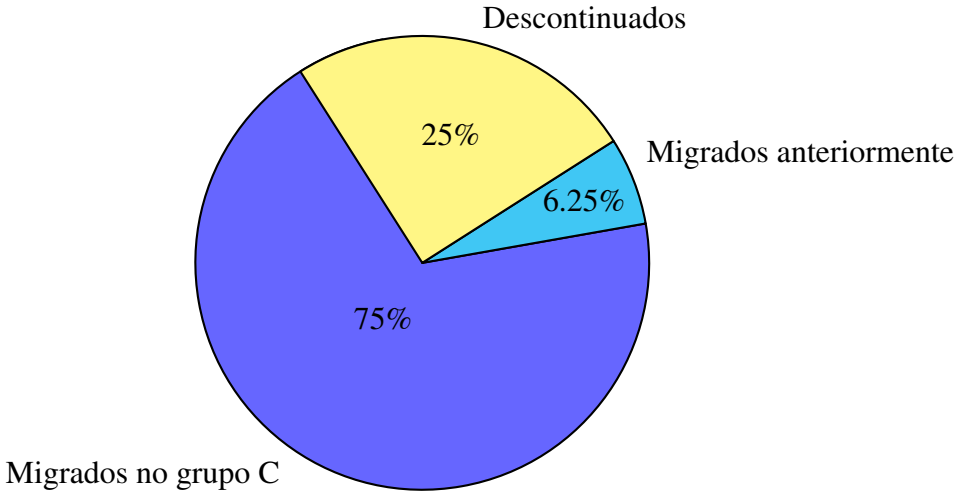

Figura 5.4: Resultados da migração do Grupo C

5.5 Grupo D

Este Grupo era maioritariamente composto por documentos que se imprimem por outras vias que não o Motor de Impressão, pelo que apenas um deles era gerado por via do MI, sendo migrado com sucesso.

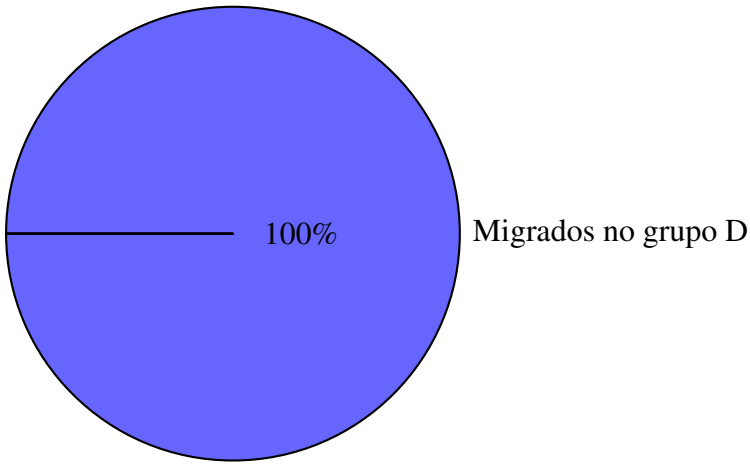

Figura 5.5: Resultados da migração do Grupo D

5.6 Grupo E

O Grupo E foi migrado com sucesso. Este era constituído por 20 documentos.

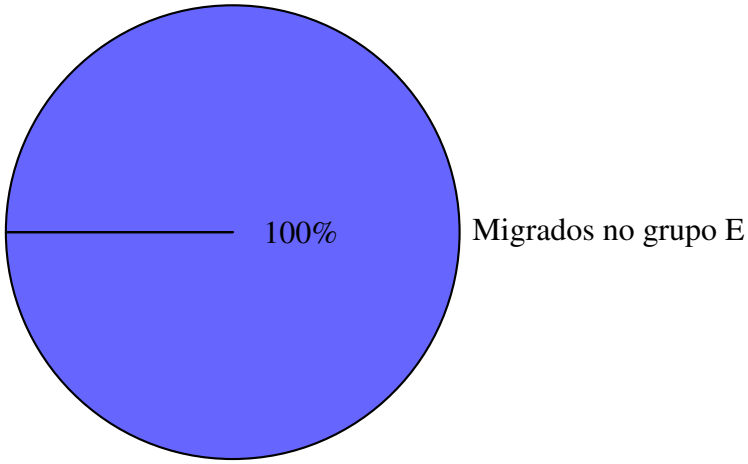

Figura 5.6: Resultados da migração do Grupo E

5.7 Grupo F

Dos 24 documentos que correspondiam a este grupo de migração, 17 deles foram migrados com sucesso, 1 já tinha sido migrado anteriormente e 6 foram descontinuados.

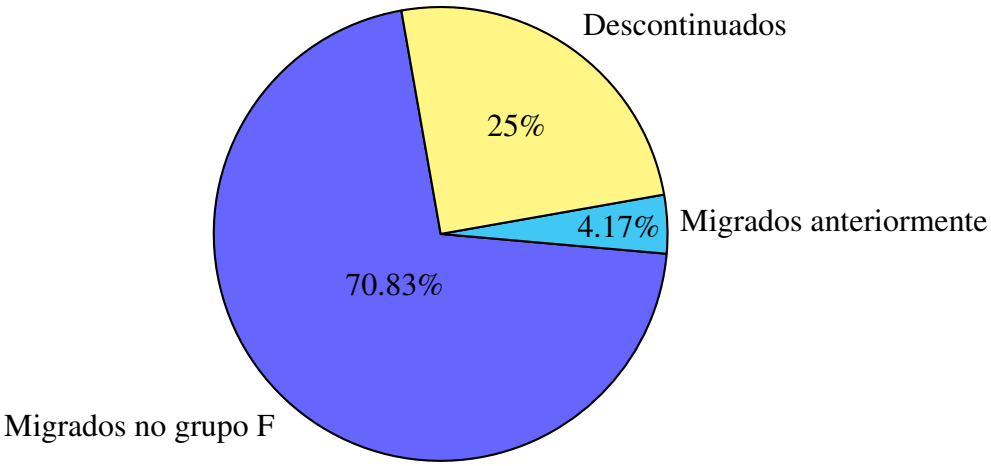

Figura 5.7: Resultados da migração do Grupo F

5.8 Discussão

Nos resultados apresentados acima, verifica-se que, dos documentos propostos a migração, 85,90% foram migrados com sucesso, sendo que os casos em que isto não ocorreu, deveu-se ao facto de serem descontinuados ou já terem sido migrados.

Esta migração permitiu assim que alguns documentos fossem descontinuados pois deixaram de ser obtidos. Neste caso, já não fazia sentido para a companhia que estes fossem migrados para o HPX, deixando assim de ser possível obtê-los quando esta ferramenta for a única em uso. Existiram ainda documentos que são apresentados nos resultados como migrados anteriormente, pois necessitaram de ser alterados e, visto que a migração era expectável, não fazia sentido a alteração ser feita no AOD para depois o seu modelo ser migrado para o HPX.

Os resultados contemplam apenas os documentos migrados e que são relativos ao Motor de Impressão. O projeto incluía tanto documentos obtidos através do MI como por outras vias dentro do sistema, daí que os grupos de migração estejam muito desproporcionais relativamente a números de documentos que passam pelo MI. Por exemplo, o grupo D contempla quase na sua totalidade documentos que não são obtidos pelo MI, daí nos resultados ser apresentado apenas um documento.

Para a seleção e divisão dos documentos nos respetivos grupos também foi utilizado o grupo (tipo) a que estes pertencem, pois para a equipa que desenvolveu os novos modelos no HPX é mais fácil migrar documentos que sejam semelhantes, principalmente pelas vantagens já referidas da nova ferramenta.

Capítulo 6

Conclusão e trabalho futuro

Com a grande corrente de transformação digital e a nova era tecnológica, as atualizações a nível de mercado e de ferramentas utilizadas na execução de tarefas nas diversas empresas devem estar em constante atualização. Por vezes, é necessária a migração destas ferramentas, de forma a melhorar os índices das empresas. Desta forma, este projeto incorpora o objetivo da constante atualização e melhoria da companhia, bem como o desenvolvimento mais rápido e otimizado de novos modelos para a geração de PDFs.

Consequentemente foi necessário atualizar a informação presente na base de dados do Motor de Impressão, bem como os seus *outputs*. Além disso, foi necessário alterar os Objetos de Dados para também estes preparem a informação em conformidade com as especificações do HPX.

O projeto teve alguns desafios mais complexos sendo eles os seguintes:

- A falta de familiarização por parte da equipa que cria os modelos para serem gerados os PDFs pois esta apenas tinha experiência na ferramenta antiga, tendo de passar por uma fase de aprendizagem já no decorrer do projeto.
- A minha necessidade de ambientação ao Motor de Impressão de forma a que pudesse realizar o projeto.
- Dada a quantidade de modelos a migrar ser muito grande, nem sempre os dados necessários existiam em Ambiente de Testes pelo que foi necessário que esses casos de teste fossem criados.
- A complexidade na obtenção de certos documentos, pois muitos deles não são muito usuais de se obter.
- A Equipa de Testes era nova portanto não tinha um grande conhecimento do sistema pelo que foi necessário dar apoio não esperado aquando do planeamento.

Todas as questões foram ultrapassadas, tendo por isso o projeto considerado um sucesso, sendo que na semana de finalização do estágio foi dada a entrada em ambiente de produção dos últimos dois grupos de documentos migrados.

Com a realização deste projeto é possível adotar uma ferramenta mais atual, permitindo à empresa uma otimização dos recursos não só humanos mas também a nível de *hardware*. Este projeto permite assim desligar os servidores onde corre a ferramenta antiga. Estes contém um sistema operativo desatualizado devido à antiguidade da ferramenta o que pode consumar-se em graves problemas de segurança para a companhia.

Após a entrada de todas as fases do projeto em ambiente de produção foi possível observar que ainda cerca de 15 documentos a ser gerados através da ferramenta antiga e que não tinham sido colocados em nenhum grupo de migração. Assim, como trabalho futuro, deve ser feita a migração dos documentos em falta de forma a que se possam desligar os servidores em que é executada a ferramenta antiga e assim, o seu uso possa ser descontinuado.

Abreviaturas

AOD Adobe Output Designer. 1–4, 20, 34, 41, 47

BD Base de Dados. 36

CGI Camada de Gestão de Interfaces. 16

CRM Customer Relationship Management. 17

DDL Data Definition Language. 34, 35

DLL Dynamic-link library. 13, 26, 35

DML Data Manipulation Language. 34

EAI Enterprise Application Integration. 16, 17, 26, 36

HPX HP Exstream. 3–6, 33, 34, 37, 39, 47, 49

HTML Hypertext Markup Language. 12, 23

IP Internet Protocol. 10

LGI Liferay Gestão de Interfaces. 16

MI Motor de Impressão. v, x, xiii, 1–6, 9, 15, 18–20, 23, 26, 27, 29, 30, 33, 35–37, 39–41, 47

MW Middleware. 36

SF SalesForce. 17

SOA Arquitetura orientada a serviços. 2, 9, 10, 16, 19

SOAP Simple Object Access Protocol. 9, 10, 16

SP Stored Procedure. 34

WSDL Web Service Description Language. 9, 11, 12, 16, 36

XML eXtensible Markup Language. ix, x, xiii, 2–6, 9–12, 16, 19–21, 25–27, 29, 33–36, 38

Bibliografia

[1] Liferay DXP — A Plataforma que Unifica a Experiência do Cliente.

[2] Wesal Al Belushi and Youcef Baghdadi. An approach to wrap legacy applications into web services. *Proceedings - ICSSSM'07: 2007 International Conference on Service Systems and Service Management*, (May), 2007.

[3] Itzik Ben-Gan, Dejan Sarka, Ed Katibah, Grew Low, Roger Wolter, and Isaac Kunen. *Inside Microsoft SQL Server 2008: T-SQL Programming*. 2010.

[4] L M Camarinha-Matos, H Afsarmanesh, E C Kaletas, and T Cardoso. Service Federation in Virtual Organizations. 2002.

[5] Kishore Channabasavaiah and Kerrie Holley. Migrating to a service-oriented architecture. Technical Report April, 2004.

[6] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Service Definition Language (WSDL), 2001.

[7] D Fensel and C Bussler. The Web Service Modeling Framework WSMF. Technical report.

[8] Antibes Jean-Marie Stawikowski and Le Thoronet Christian Hardy. COMMUNICATION SYSTEM OF AN AUTOMATION EQUIPMENT BASED ON THE SOAP PROTOCOL, 2008.

[9] Michel Klein. XML, RDF, and Relatives. Technical report, 2001.

[10] Gunnar Mein, Shankar Pal, Govinda Dhondu, Thulusalamatom Krishnamurthi Anand, Alexander Stojanovic, Mohsen Al-Ghosein, and Paul M. Oeuvray. SIMPLE OBJECT ACCESS PROTOCOL, 2006.

[11] Microsoft. What is a DLL?, 2018.

[12] Mike P Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. Technical report.

[13] Paul Turley and Dan Wood. *T-SQL*. 2008.